Janne Kauppinen

Linssit ohjelmoinnissa

Tietotekniikan kandidaatintutkielma

3. kesäkuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Janne Kauppinen

Yhteystiedot: janne.a.kauppinen@student.jyu.fi

Työn nimi: Linssit ohjelmoinnissa

Title in English: Lenses in programming

Työ: Kandidaatintutkielma

Sivumäärä: 27+0

Tiivistelmä: Ohjelmoinnissa on usein tilanteita, joissa kaksi toisiinsa yhteyksissä olevaa rakennetta on sidoksissa toisiinsa niin, että muutokset yhteen rakenteeseen heijastuvat asianmukaisesti myös toiseen rakenteeseen. Tässä tutkielmassa käsitellään datamallien välistä transformointia kaksisuuntaisten transformaatioiden näkökulmasta. Tekstissä käydään läpi erityisen tarkasti eräs kaksisuuntainen transformaatio, nimeltään linssit, ja tutkitaan linssien rakenteita ja ominaisuuksia. Linssien yhteydessä käydään läpi alunperin relaatiotietokannoista tullut termi näkymänpäivitysongelma, joka on yleistettävissä datamallien välisiin transformaatioihin. Linssit ovat vielä tällä hetkellä melko harvinaisia ohjelmoinnissa, mutta nykyään on olemassa ohjelmointikieliä, jotka perustuvat kokonaan linsseihin. Lisäksi Haskelissa on tarjolla laaja linssikirjasto.

Avainsanat: linssit, kaksisuuntainen transformaatio, näkymänpäivitysongelma, Haskell

Abstract: Computing is full of situations where two structures are connected in such a way that modification to one structure should be propagated to the other structure. This reseach is about bidirectional transformations between data structures. This text focuses on a special case of bidirectional transformations called lenses. The structure and the properties of lenses is studied in a general but still in a detailed way. The text also covers a term called view update problem which has been extensively studied in context of relational databases. The data transformations between two different structures is a generalization of the view update problem. Lenses are still rare in the context of programming but there are, however, programming languages which are based on lenses. Haskell has also a large implementation of lens library.

Keywords: lens, bidirectional transformation, view update problem, Haskell

Kuviot

Sisältö

1 Johdanto

Ohjelmoinnissa on usein tilanteita, joissa kaksi erilaista mutta sisällöltään yhtäpitävää rakennetta on sidoksissa toisiinsa siten, että muutokset yhteen rakenteeseen heijastuvat asianmukaisesti myös toiseen rakenteeseen (Pierce 2012). Klassinen esimerkki tällaisesta tilanteesta on relaatiotietokannat ja niistä kyselyillä johdetut näkymät, joissa näkymää muokatessa myös sitä vastaava relaatiotietokanta muuttuu johdonmukaisesti.

Tällaisia datamallien välisiä transformaatioita voidaan toteuttaa käytännössä kahdella eri tavalla: Voidaan toteuttaa kaksi erillistä transformaatiota jollakin sopivaksi havaitulla ohjelmointikielellä, yksi kumpaankin suuntaan, ja varmistaa manuaalisesti että datamallien väliset transformaatiot toimivat keskenään oikein (Stevens 2008). Tämä toteutustapa ei ole käytännössä kovinkaan suotavaa, sillä kahden erillisen transformaation johdonmukaisuuden tarkistaminen on vaikeaa, virhealtista ja aiheuttaa todennäköisesti haasteita transformaatioiden ylläpidossa (Stevens 2008). Toinen tapa toteuttaa kahden datamallin väliset muunnokset on niin sanotut kaksisuuntaiset transformaatiot (engl. *bidirectional transformations*).

Olemassa olevat kaksisuuntaiset transformaatiot eroavat toisistaan monin eri tavoin, mutta yhteistä näille on se, että ne koostuvat kahdesta yksisuuntaisesta transformaatiosta, joista toinen toteuttaa tietomallin muunnoksen yhteen suuntaan, ja toinen transformaatio taas toteuttaa muutoksen päinvastaiseen suuntaan (Czarnecki ym. 2009). Yksi- ja kaksisuuntaisten transformaatioiden oleellisin ero on siinä, että kaksisuuntaisissa transformaatioissa nämä yksisuuntaiset transformaatiot kapseloidaan jollakin tavalla yhteen muodostaen siten oman yksittäisen ohjelman, jota voi ajaa kumpaankin suuntaan. Eräs tässä tekstissä tarkemmin kuvattu kaksisuuntainen transformaatio tunnetaan nimeltä linssi (engl. *lens*).

Linssit ovat tällä hetkellä harvinaisia ohjelmoinissa, mutta esimerkiksi Boomerang on ohjelmointikieli, joka on suunniteltu nimenomaan linssien käyttöön (J. N. Foster 2010). Lisäksi C-kielellä toteutettu linuxin konfiguraatiokirjasto, Augeas, perustuu linssien käyttöön ja teoriaan (Lutterkort 2007). Myös Haskell-ohjelmointikielellä on toteutettu linssikirjastoja, joista tunnetuin on Edward Kmettin vuonna 2012 julkaisema linssikirjasto (Kmett 2012).

Tässä tutkimuksessa käsitellään kaksisuuntaisia transformaatioita ohjelmoinnin näkökulmas-

ta, ja erityisen tarkasti käydään läpi kaksisuuntaiset transformaatiot nimeltään linssit. Tekstissä on kolme pääkäsittelylukua. Luvussa 2 käsitellään yleisesti kaksisuuntaisia transformaatioita ja selitetään linsseihin kehitykseen vahvasti liittyvä ilmiö nimeltään näkymänpäivitysongelma. Luvussa 3 käydään läpi linsseihin liittyviä teoria-asioita ja linssien karakterisointia. Luvussa 4 tarkastellaan lyhyesti linssejä Haskelin näkökulmasta.

2 Linsseihin liittyviä määritelmiä ja käsitteitä

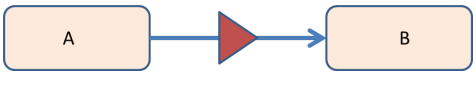
Tässä luvussa käsitellään linsseihin vahvasti liittyvät käsitteet: kaksisuuntainen transformaatio ja näkymänpäivitysongelma (engl. *view update problem*).

2.1 Kaksisuuntainen transformaatio

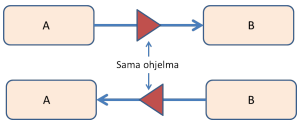
Merkitään kahta eri datamallia kirjaimilla A ja B. Yksisuuntainen transformaatio (engl. *unidirectional transformation*) (ks. kuvio 1a) voidaan kuvata merkinnällä f : A -> B, missä f on yksisuuntainen transformaatio datamallista A datamalliin B. Ohjelmoinnin näkökulmasta f tarkoittaa ohjelmaa, jota voi suorittaa yhteen suuntaan ottaen argumenttina A:n ja palauttaa ulostulona B:n. Tässä tekstissä sanalle ohjelma ei anneta tarkkaa määritelmää, vaan ohjelmalla tarkoitetaan yleisesti ohjelmakoodia, kuten esimerkiksi funktioita.

Kaksisuuntainen transformaatio (ks. kuvio 1b) on mekanismi kahden tai useamman informaatiolähteen johdonmukaisuuden ylläpitämiseksi (Czarnecki ym. 2009). Kahden informaatiolähteen välinen kaksisuuntainen transformaatio koostuu kahdesta yksisuuntaisesta transformaatioista: eteenpäin transformaatiosta (engl. *forward transformation*) A:sta B:hen, ja taaksepäin transformaatiosta (engl. *backward transformation*) B:stä A:han (Czarnecki ym. 2009). Taaksepäin transformaatiota kutsutaan myös nimellä käänteinen transformaatio (Czarnecki ym. 2009).

Yleensä datamallien väliset transformaatiot toteutetaan määrittämällä kaksi erillistä yksisuuntaista ohjelmaa, mutta tällä tavalla esitettynä transformaatiot ovat vaikeita implementoida, ymmärtää ja erittäin hankalia ylläpitää (J. N. Foster 2010). Vaihtoehtoinen ja parempi tapa toteuttaa kaksisuuntainen transformaatio on määritellä se yhtenä ohjelmana, jota voi



(a) Yksisuuntainen transformaatio.



(b) Kaksisuuntainen transformaatio.

Kuvio 1: Kuvat yksi- ja kaksisuuntaisesta transformaatiosta.

ajaa sekä eteen- että taaksepäin, tai toisella tavoin ajateltuna, vasemmalta oikealle ja oikealta vasemmalle (J. N. Foster 2010). Tällaista ohjelmaa, jota voi suorittaa kumpaankin suuntaan, kutsutaan kaksisuuntaiseksi transformaatioksi.

Datavirta A:sta B:hen dominoi usein datavirtaa B:stä A:han siten, että data A:n sanotaan olevan kaksisuuntaisen transformaation syöte, lähdedata tai isäntä, ja B:tä sanotaan kaksisuuntaisen transformaation ulostuloksi, kohdedataksi tai orjaksi (Czarnecki ym. 2009). Tämä tarkoittaa käytännössä sitä, että kohdedatamalli on suora abstraktio lähdedatamallista: kohdemalli on sisältämänsä informaation suhteen lähdemallin osajoukko (Stevens 2007). Esimerkki tällaisesta tilanteesta on muun muassa relaatiotietokanta ja siitä kyselyllä johdettu näkymä. Tällöin siis relaatiotietokanta on kaksisuuntaisen transformaation lähdedata ja näkymä on puolestaan kohdedata. Relaatiotietokanta asettaa ehtoja sille, millaisia näkymiä siitä voidaan johtaa. Tällaista kaksisuuntaista transformaatiota, jossa on toinen rakenne on toisen rakenteen abstraktio, kutsutaan asymmetriseksi kaksisuuntaiseksi transformaatioksi.

Toisinaan on tilanteita, jossa kahdella rakenteella voi sama informaatio, mutta vain eri muodossa, ja muutokset toiseen datarakenteeseen tulisi heijastua asianmukaisesti myös toiseen datarakenteeseen (Wagner 2014). Tällaista kaksisuuntaista transformaatiota, jossa kumpikin rakenne sisältää eri muodossa olevan saman datan, kutsutaan symmetriseksi kaksisuuntaiseksi transformaatioksi (Wagner 2014). Ohjelman lähdekoodi ja siitä jäsennetty abstrakti syntaksipuu on yksi esimerkki symmetrisestä tilanteesta.

Helpoin tapa esittää kaksisuuntainen transformaatio teksteissä tai kaavioissa on esittää se funktioparina: yhtenä funktiona kutakin suuntaa kohden (Stevens 2008). Stevensin mukaan tällöin voidaan välttyä kaksisuuntaisia transformaatioita varta vasten suunniteltujen ohjelmointikielten määrittämiseltä, ja olemassa olevia ohjelmointikieliä voidaan käyttää kaksisuuntaisten transformaatioiden esittämiseen. Tosin on olemassa myös sellaisia ohjelmointikieliä, joissa jokainen lauseke on kaksisuuntainen transformaatio (Pierce 2012). Tällaisia ohjelmointikieliä kutsutaan kaksisuuntaisiksi ohjelmointikieliksi (J. N. Foster 2010).

Erityisen tärkeä huomio kaksisuuntaisissa transformaatioissa on se, että niiden ei tarvitse olla bijektiivisiä transformaatioita (Stevens 2008). Vaikka bijektiivisiä kaksisuuntaisia transformaatioita tarvitaan, katsotaan bijektiivisyyden olevan usein liian vahva ja jopa ei-toivottu

ominaisuus. Esimerkiksi jos ajatellaan ohjelman lähdekoodia ja siitä käännettyä ohjelmaa datamalleina, niin kääntäjä voi optimoida kahdesta, toisistaan hieman poikkeavista, lähdekoodeista saman lopputuloksen (Stevens 2007). Tai jos xml-dokumentissa on merkityksettömiä tyhjiä merkkejä (engl. *white spaces*), niin niitä ei ole järkevää ottaa mukaan lopulliseen dokumenttiin ikään kuin ne olisivat merkityksellisiä (J. N. Foster 2010).

Seuraavaksi tarkastellaan ilmiötä nimeltään näkymänpäivitysongelma, joka liittyy vahvasti kaksisuuntaisten tranformaatioiden ja erityisesti linssien kehityksen taustaan (Aaron Bohannon ja Vaughan 2006).

2.2 Näkymänpäivitysongelma

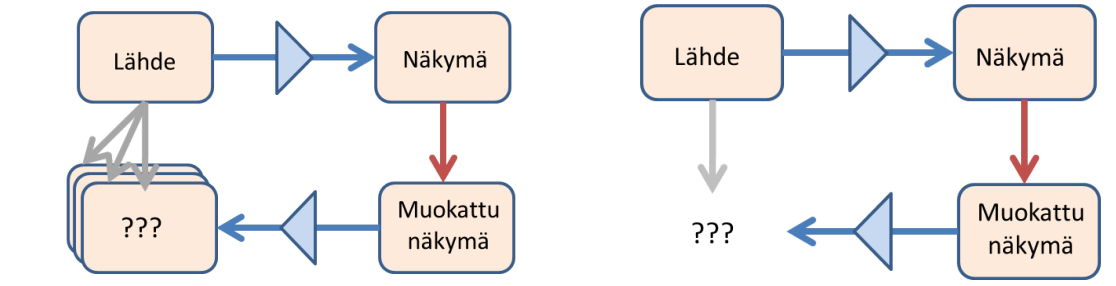
Linssien kehitys liittyy vahvasti kaksisuuntaisiin transformaatioihin sekä niin sanottuun näkymänpäivitysongelmaan. Näkymänpäivitysongelma liittyy alunperin relaatiotietokantoihin (Bancilhon ja Spyratos 1981), mutta on kuitenkin hyvin yleinen ilmiö monissa eri sovellusalueissa (J. N. Foster 2010). Seuraavaksi kuvataan mistä alkuperäisessä näkymänpäivitysongelmassa on kyse.

Relaatiotietokannoissa data on yleensä hyvin kompleksisessa muodossa, ja niistä on mahdollisuus hakea dataa tietokantakyselyillä (engl. *database query*). Kyselyn lopputuloksena saadaan näkymä haettuun dataan (engl. *view*), mutta pelkkä näkymän hakeminen tietokannasta ei aina riitä, vaan usein halutaan myös muokata tietokannan tietoja näkymän kautta.

Näkymänpäivitysongelmaan kuuluu kaksi perustapausta (ks. kuvio 2). Ensimmäinen perustapaus on se, että usein näkymän muokkauksella ei ole yksikäsitteistä vastaavaa muokkausoperaatiota tietokannassa (Bohannon, Pierce ja Vaughan 2006). Näkymän muokkauksella voi olla siis useita eri muokkausvaihtoehtoja tietokannassa, jolloin ongelmaksi muodostuu oikean vaihtoehdon valinta (Bancilhon ja Spyratos 1981).

Toinen potentiaalinen ongelmatilanne on se, että näkymän muokkauksella ei välttämättä ole olemassa vastaavaa muokkausoperaatiota tietokannassa (J. N. Foster 2010). Tämä tarkoittaa käytännössä sitä, että näkymään tehtyjä muokkauksia ei voi määritellä tietokannassa.

Näkymänpäivitysongelmiin on kehitetty erilaisia ratkaisuja. Esimerkiksi siinä tilanteessa,



(a) Lähdedatassa useita eri päivitysvaihtoehtoja.

(b) Lähdedatassa ei päivitysvaihtoehtoja.

Kuvio 2: Näkymänpäivitysongelman perustapaukset.

jossa mahdollisia päivitysvaihtoehtoja on useita, voidaan asettaa ylimääräisiä rajoitteita, joiden avulla voidaan ohjata ongelmanratkaisua kohti oikeaa päivitysvaihtoehtoa siten, että muutokset lähdedatassa olisivat mahdollisimman pieniä (J. N. Foster 2010). Fosterin mukaan tällöin sellaisten päivitysten laskeminen, jotka toteuttavat asetetut rajoiteet, tulevat haasteelliseksi, jos kyselykieli tai tietokannan skeema muuttuu.

Tilanteessa, jossa näkymän päivityksellä ei ole lainkaan vastaavuutta tietokannassa, voidaan järjestelmä saada hylkäämään kyseenomaiset päivitykset (J. N. Foster 2010). Järjestelmään asetetaan piilossa olevia rajoitteita, jotka määräävät sen kuinka näkymä voidaan päivittää. Foster sanoo väitöskirjassaan, että tällaisissa tapauksissa ongelmaksi nousevat nämä piilevät rajoitteet, niin sanotut vuotavat abstraktiot, jotka muuten olisivat piilossa käyttäjältä, mutta nousevat esille, kun käyttäjä yrittää suorittaa näkymän päivitystä tietokantaan.

Edellämainituista syistä näkymät ovat usein luonteeltaan vain lukuoperaatioita, ja sellaisissa tapauksissa, joissa halutaan toteuttaa näkymän kautta suoritettavia tietokantapäivityksiä, on ohjelmoijan määriteltävä niin sanottuja triggereitä, jotka laukaistaan, kun näkymää muokataan (J. N. Foster 2010). Triggeri on proseduuri, joka määrittää miten tietty näkymänpäivitys siirretään tietokantaan (esimerkiksi joukko sql-käskyjä), ja näillä triggereillä voidaan ohjelmoida jokainen järkeenkäypä näkymänpäivitys. Tämän tavan varjopuoli on se, että ohjelmoijan on manuaalisesti testattava jokaisen triggerin toiminta ja, jos tietokannan skeema muuttuu, triggereiden ylläpito muuttuu vaikeaksi (J. N. Foster 2010).

Näkymänpäivitysongelma ei rajoitu vain relaatiotietokantoihin, vaan se on hyvin yleinen ongelma datan transformoinneissa. Yksi tapa lähestyä näkymänpäivitysongelmaa yleisellä ta-

solla ovat kaksisuuntaiset transformaatiot nimeltään linssit. Linsseissä sekä näkymän määrittäminen että päivitystoimintaperiaate kapseloidaan yhteen, ja tämän seurauksena ohjelmoijan tarvitsee toteuttaa ja ylläpitää ohjelmakoodia ainoastaan yhdessä paikassa (J. N. Foster 2010).

3 Linssien teoriaa

Tässä kappaleessa esitetään linssin määritelmä sekä linsseihin liittyvää terminologiaa. Lisäksi käydään läpi linsseihin liittyviä vaatimuksia, joita linsseiltä yleensä vaaditaan, ja niitä sääntöjä, joita linssien täytyy toteuttaa jotta linssit pystyvät toteuttamaan halutut vaatimukset.

3.1 Linssin määritelmä

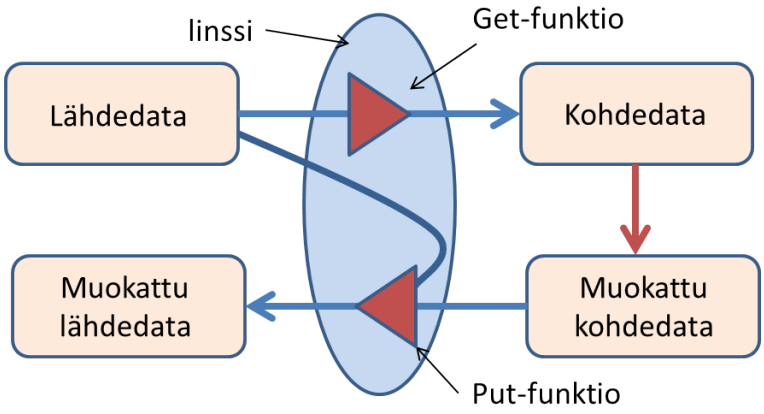
C. Pierce ja muut esittelivät linssin käsitteen yleisesti vuonna 2007 (Foster ym. 2007). Linssi on kaksisuuntainen transformaatio, joka määrittelee kaksi funktiota, jolla voidaan kuvata yhteen rakenteeseen tehdyt muutokset myös toiseen rakenteeseen (Pierce 2012).

Linsseissä funktiot on nimetty seuraavasti (ks. kuvio 3): eteenpäin transformaatio on nimeltään get, ja taaksepäin transformaatio on nimeltään putback (Foster ym. 2007). Tosin yleensa putback-funktiota kutsutaan lyhyemmin nimellä put, ja tässä tekstissä käytetään jatkossa funktion lyhennettyä nimeä.

Yleensä linssejä käytetään asymmetrisissä tilanteissa eli silloin, kun toinen rakenne on selvästi primäärinen ja toinen on abstraktio primäärirakenteesta, mutta myös linssien käyttöä symmetrisissä tapauksissa on tutkittu (Wagner 2014). Tässä tekstissä keskitytään käsitelemään lähinnä asymmetrisiä linssejä.

Määritellään ensin get-funktio. Get-funktio on se linssiin kuuluva funktio, joka ottaa argumettina lähdedatan, ja palauttaa näkymän lähdedatasta. Put-funktio on tyypiltään hieman erilainen verrattuna get-funktioon. Put-funktio on funktio, joka laskee muokatusta näkymästä uuden lähdedatan. Put-funktio eroaa get-funktiosta myös siinä mielessä, että se ottaa argumentteinään kaksi arvoa: muokatun näkymän ja alkuperäisen lähdedatan.

Syy siihen, miksi put-funktio ottaa argumenttinä myös alkuperäisen lähdedatan, on se, että get-funktio hävittää yleensä osan alkuperäisestä datasta laskiessaan näkymää (J. N. Foster 2010). Linssien kohdalla datarakenteilla on yleensä keskenään asymmetrinen rakenne, joten näkymä on tällöin abstraktio lähderakenteesta. Näkymä ei siis yleensä sisällä kaikkea sitä



Kuvio 3: Graafinen esitys linssistä (mukaillen N. Foster 2010).

tietoa jota tarvitaan uuden lähdedatan muodostamiseksi. Voidakseen rakentaa uuden päivitetyn lähdedatan, put-funktion täytyy saada muokatun näkymän lisäksi myös ne näkymästä puuttuvat lähderakenteen tiedot, joita tarvitaan uuden lähdedatan rakentamiseksi. Tässä sekä get- että put-funktiot on esitetty Haskel-ohjelmointikielen syntaksin mukaisesti (Lipovaca 2011). Lähderakenteen tyyppiä on merkitty kirjamella S ja lähderakenteesta lasketun näkymän tyyppiä on merkitty V:llä.

```
get :: S -> V
put :: V -> S -> S
```

John Foster esittelee väitöskirjassaan myös create-funktion, joka on samankaltainen put-funktion kanssa, mutta ottaa argumenttina ainoastaan näkymän ja palauttaa lähteen. Tämä funktio on sellaisia erikoistapauksia varten, joissa lähdettä ei ole saatavilla, vaan uusi lähde täytyy luoda näkymästä siten, että puuttuvat tiedot korvataan oletusarvoilla (J. N. Foster 2010). Tässä tutkielmassa ei käsitellä tarkemmin create-funktiota, vaan tässä keskitytään linssien yleisimpiin funktioihin, eli get- ja put-funktioon.

Linssit jaetaan put-funktion määrittelyn mukaan joko tila- tai operaatioperusteisiin linsseihin. Jos put-funktio saa argumenttinaan ainoastaan päivitetyn lähdedatan, niin tällöin kyseessä on tilaperusteinen linssi (J. N. Foster 2010). Jos put-funktion argumenttinä saatu näkymä sisältää myös ylimääräistä tietoa siitä, miten näkymää tulisi transformoida lähdedataan, niin tällöin linssiä sanotaan operaatioperusteiseksi linssiksi (J. N. Foster 2010). Tässä tekstissä

käsitellään ainoastaan tilaperusteisia linssejä.

Linssit vaativat yleensä ohjelmointikieleltä hyvin ilmaisuvoimaisen tyyppijärjestelmän (Pierce 2012). Jotta linssillä voitaisiin toteuttaa monimutkaisia transformaatioita lähde- ja kohdedatan välillä, täytyy linssin olla sellaista tyyppiä, että se pystyy kuvaamaan sekä lähde- että kohderakenteet korkealla tarkkuustasolla (J. N. Foster 2010). Tämän lisäksi tarkka tyyppijärjestelmä mahdollistaa linssien yhdistämisen toisiinsa (J. N. Foster 2010). Linssejä voidaan siis yhdistää peräkkäin niin sanottujen kombinaattoreiden avulla. Kombinaattorit voivat vaihdella kontekstista riippuen, mutta esimerkiksi Boomerang-ohjelmointikielessä kombinaattoreita ovat ketjuttaminen, unioni ja Kleenen tähti (J. N. Foster 2010). John Fosterin mukaan Boomerang-ohjelmointikielessä linssejä voidaan ketjuttaa komposition avulla, unioni-operaation avulla voidaan muodostaa ehtoja linsseille ja Kleenen tähti-operaatiolla voidaan yhdistää edellämainittuja kombinaattoreiden käytöksiä. Tällä tavalla yksinkertaisista, niin sanotuista primitiivilinsseistä, voidaan rakentaa isompia ja monimutkaisempia linssejä (J. N. Foster 2010).

3.2 Linssien algebralliset ominaisuudet

Seuraavaksi käydään läpi asioita, joita tulee huomioida, jotta linssit toteuttaisivat transformaatiot oikein. Se, että linssi toteuttaa get- ja put-funktiot, ei anna juurikaan tietoa siitä, miten linssit käyttäytyvät. Jotta linssit voisivat toteuttavaa datan transformaation oikein, täytyy get- ja put-funktioille asettaa rajoituksia ja sääntöjä siitä, miten nämä funktiot toimivat keskenään. Erilaiset rajoitukset antavat linsseille toisistaan poikkeavia ominaisuuksia, jolloin voidaan valita eri tilanteisiin oikeanlaiset linssit.

Linssit voidaan jaotella eri linssiluokkiin riippuen siitä, miten get- ja put-funktiot käyttäytyvät keskenään (Fischer, Hu ja Pachero 2015). Linssien täytyy täyttää tietyt ehdot ennen kuin ne voivat muodostaa päivitetystä näkymästä oikeanlaisen lopputuloksen. Yleisesti linssit on suunniteltu takaamaan seuraavat kolme asiaa (J. N. Foster 2010)

1. Linssin käyttäjä voi tehdä mielivaltaisia muutoksia näkymään välittämättä siitä, ovatko nämä muutokset yhteensopivia lähdedatan kanssa. 2. Linssit saattavat päivitykset yksikäsitteisesti lähdedataan. 3. Aina kun mahdollista, linssit säilyttävät kaiken sen lähdeinformaation,

johon näkymän muutokset eivät heijastu.

Jotta edellämainitut ominaisuudet voidaan toteuttaa, vaaditaan yleensä, että put-funktion täytyy olla täydellinen funktio ja ensimmäisen argumenttinsä suhteen injektiivinen (J. N. Foster 2010). Täydellisellä funktiolla tarkoitetaan sellaista funktiota, joka on määritelty kaikilla sen laillisilla syötteillä. Tämä ominaisuus takaa sen, että linssit pystyvät tekemään jotain järkevää jokaisella näkymällä ja lähteellä jopa silloin, kun näkymää muutetaan dramaattisesti (J. N. Foster 2010). Linssit vaativat ohjelmointikieleltä erittäin tarkan tyyppijärjestelmän, sillä näkymän varma päivittäminen varmistetaan näkymän tyypin perusteella (J. N. Foster 2010). Tarkka tyyppijärjestelmä mahdollistaa kohderakenteen tarkan määrittämisen, jolloin voidaan määritellä put-funktiolle hyvin tarkasti millaisen näkymän se voi ottaa argumenttina. Hyvin yksityiskohtainen tyyppijärjestelmä, put-funktion täydellisyys ja semi-injektiivisyys antavat edellytykset luotettavaan datan transformaatioon.

Luotettavaan transformaatioon vaaditaan myös se, että sekä get- että put-funktiot toimivat järkevästi keskenään. Linsseille on kehitetty erilaisia lakeja, jotka takaavat linsseille tietynlaisen käyttäytymisen (Foster ym. 2007). Tässä esitellään yleisimmät rajoitteet ja ominaisuudet, joita linsseiltä yleensä vaaditaan. Määritellään niin sanotut GetPut-, PutGet- ja PutPut-lait. Merkitään s:llä lähdettä ja v:llä näkymää. Olkoon s $\in$ S ja v' ja v $\in$ V.

```
put (get s) s = s                (GETPUT)
get (put v s) = v                (PUTGET)
put v' (put v s)) = put v' s     (PUTPUT)
```

GetPut-laki kertoo sen, että jos lähdedatasta lasketaan get-funktiolla näkymä, ja tämän jälkeen lasketaan put-funktiolla muuttamaton näkymä takaisin alkuperäiseen lähteeseen, on lopputulos sama kuin alkuperäinen lähde (Foster ym. 2007). Toisin sanoen, jos näkymä ei muutu, niin lähteenkään ei tule muuttua. Yleensä halutaan, että put-funktio ei tee lähdedataan tarpeettomia muutoksia. Toisinaan transformaatio vaatii kuitenkin sen, että put-funktiolla täytyy olla sivuvaikutuksia siihenkin osaan lähdedataa, joka ei ole mukana näkymässä (J. N. Foster 2010). Foster sanoo väitöskirjassaan, että GetPut-laki on yksi mahdollisista ehdoista, joilla voidaan kontrolloida lähdedatan koskemattomuutta. GetPut-laki ei pysty yksinään takaamaan lähdedatan koskemattomuutta, mutta se on käyttökelpoinen linssien suunnittelussa:

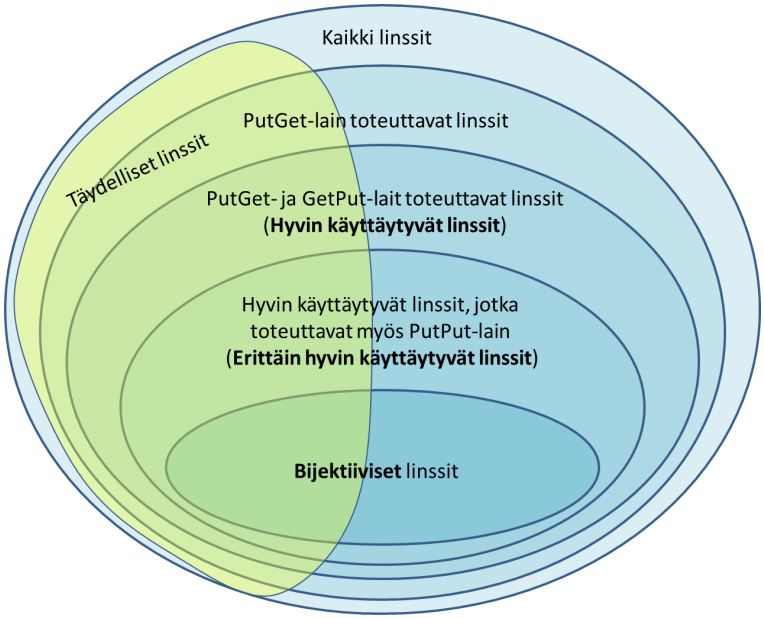
GetPut-lain avulla voidaan luoda, testata ja hylätä linssikandidaatteja (J. N. Foster 2010).

PutGet-laki kertoo puolestaan sen, että jos ensin lasketaan put-funktiolla näkymä v lähteeseen s, ja tämän jälkeen haetaan get-funktiolla saadusta lähteestä näkymä, niin lopputulokseksi täytyy saada sama alkuperäinen näkymä v. Tämä laki takaa sen, että put-funktio ei muuta näkymän sisältämää dataa, vaan päivittää näkymän lähdedataan täsmällisesti (Foster ym. 2007). PutGet-laista seuraa se, että put-funktion täytyy olla ensimmäisen argumenttinsa suhteen injektiivinen funktio. (J. N. Foster 2010).

Toisinaan halutaan, että put-funktio tekee ainoastaan näkymässä olevia muutoksia lähdedataan, ja jättää muun lähdedatan koskemattomaksi (J. N. Foster 2010). Tällaisessa tapauksessa linssin tulee noudattaa PutPut-lakia. PutPut-laki takaa sen, että jos saman linssin put-funktiota kutsutaan peräkkäin, niin viimeisin put-funktion tulos jää voimaan (Foster ym. 2007). Toisin sanoen uusin put-funktion kutsu korvaa edelliset put-funktiot. Lisäksi PutPut-laki takaa sen, että put-funktio ei muuta näkymän ulkopuolella olevaa lähdedataa (J. N. Foster 2010).

PutPut-laki takaa siis sen, että put-funktio säilyttää kaiken sen lähdedatan koskemattomana, joka ei heijastu näkymään. Samaa asiaa on tutkittu relaatiotietokantojen parissa, ja siellä sama idea tunnetaan nimellä vakiokomplementtiehto (engl. *constant complement condition*) (Bancilhon ja Spyratos 1981). Vakiokomplementtiehdossa on ajatuksena se, että lähdeen S tulisi olla isomorfinen suhteessa pariin (V,C), missä V on lähteestä saatu näkymä ja C sisältää kaiken sen lähdedatan informaation, joka ei heijastu näkymään (J. Nathan Foster ja Zdancewic 2009). Get-funktio käyttää isomorfista kuvausta hyväkseen laskeakseen lähteestä S parin (V,C), jonka jälkeen get-funktio hävittää komplementin C ja ottaa käyttöönsä näkymän V. Put-funktio luo muokatusta näkymästä V' ja komplementista C uuden parin (V',C), ja käyttää isomorfista kuvausta toiseen suuntaan kuvatakseen uuden parin uudeksi lähdedataksi (ks. kuvio 7) (J. Nathan Foster ja Zdancewic 2009).

Sellaisia linssejä, jotka toteuttavat sekä GetPut- että PutGet-lait, kutsutaan hyvin käyttäytyviksi linsseiksi (engl. *well-behaved lenses*) (Foster ym. 2007). Hyvin käyttäytyvät linssit heijastavat lähteen muutokset näkymään, mutta eivät tee mitään muutoksia, jos näkymä ei muutu (Fischer, Hu ja Pachero 2015). Jos linssi toteuttaa ainoastaan PutGet-lain, niin put-

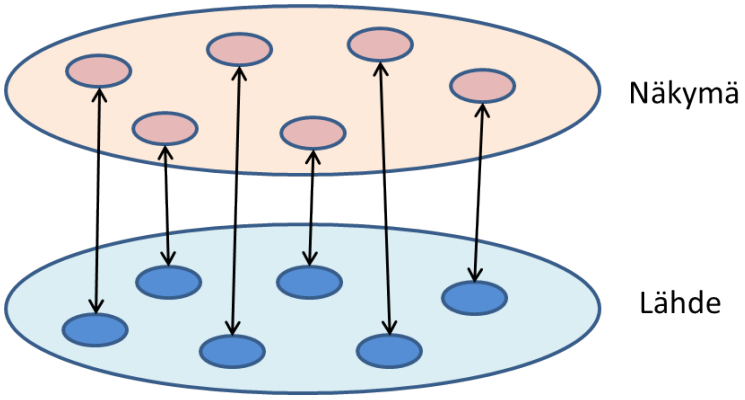


Kuvio 4: Graafinen kuva yleisimmistä linssityypeistä ja niiden hierarkisesta rakenteesta (mukaillen N. Foster 2010).

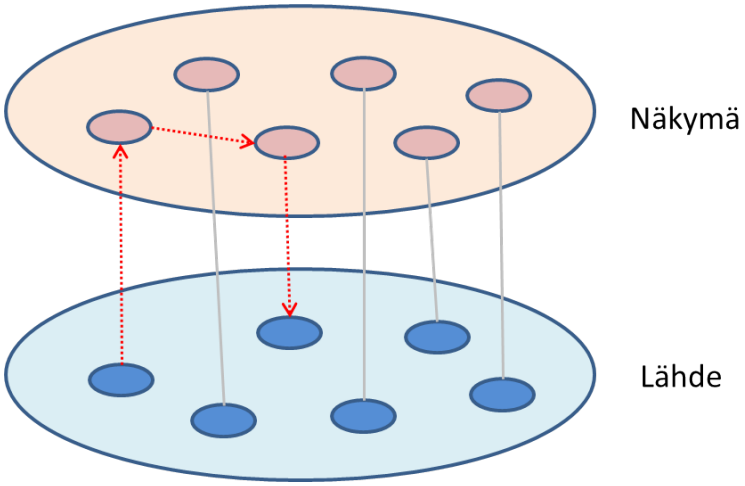
funktion toiminta ei ole niin rajoittunutta kuin siinä tapauksessa, että linssi toteuttaisi myös PutPut-lain. Tällaisessa tapauksessa, jossa linssi toteuttaa ainoastaan PutGet-lain, erilaisia put-funktiokandidaatteja voi olla useampi, jolloin oikean put-funktion valitsemiseen tarvitaan ylimääräistä informaatiota (ks. kuvio 8). Yleisesti ottaen hyvin käyttäytyvillä linsseillä get-funktiota kohden voi olla useita eri put-funktio vaihtoehtoja.

Jos linssi toteuttaa GetPut- ja PutGet-lakien lisäksi vielä myös PutPut-lain, niin tällöin sanotaan, että linssi on erittäin hyvin käyttäytyvä linssi (engl. *very well behaved lens*) (Foster ym. 2007). Erittäin hyvin käyttäyttyvän linssin put-funktion toiminta on rajoitetumpi ja tarkemmin määritelty kuin hyvin käyttäytyvien linssien kohdalla (ks. kuvio7). Erittäin hyvin käyttäytyvät linssit käyttäytyvät samankaltaisesti kuin hyvin käyttäytyvät linssit, mutta esimerkiksi get-funktio säilyttää ainoastaan näkymän tiedot ja hävittää kaiken muun lähdedatan tiedoista (Fischer, Hu ja Pachero 2015).

Kaksi ensimmäistä lakia ovat erityisen tärkeitä linssien ominaisuuksien kannalta, joten jomman kumman ominaisuuden poistaminen heikentää merkittävästi linssien semanttisia perusteita (Foster ym. 2007). Usein PutPut-lakia pidetään liian ankarana, mutta kontekstista riippuen sitä voidaan tarvita (J. N. Foster 2010). Edellämainittujen linssien lisäksi on olemassa



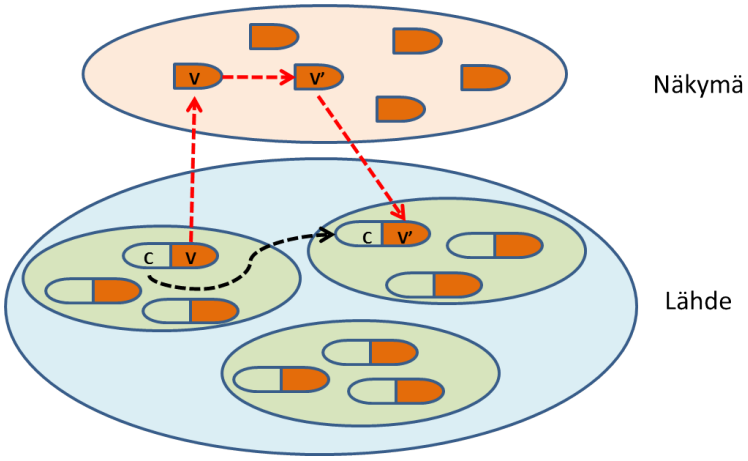
Kuvio 5: Bijektiivinen linssi määrittelee 1:1 vastaavuuden lähteen ja näkymän välille (mukaillen N. Foster 2010).



Kuvio 6: Bijektiivisen linssin kohdalla get-funktio määrää täysin put-funktion käyttäytymisen (mukaillen N. Foster 2010).

vielä bijektiiviset linssit, jotka toteuttavat kaikki edellä mainitut lait, ja jossa sekä get- että put-funktiot ovat molemmat bijektiivisiä funktioita (J. N. Foster 2010). Bijektiivisten linssien kohdalla linssit ovat bijektiivisyydestä johtuen symmetrisiä, sillä lähdeen ja näkymän välillä vallitsee yhden suhde yhteen vastaavuus (ks. kuvio 5) (Fischer, Hu ja Pachero 2015). Bijektiivisten linssien kohdalla put-funktio ei tarvitse argumentikseen alkuperäistä lähdettä, vaan se pystyy rakentamaan lähdedatan suoraan näkymästä (Fischer, Hu ja Pachero 2015). Bijektiivisten linssien kohdalla get-funktio määrää put-funktion käyttäytymisen 6.

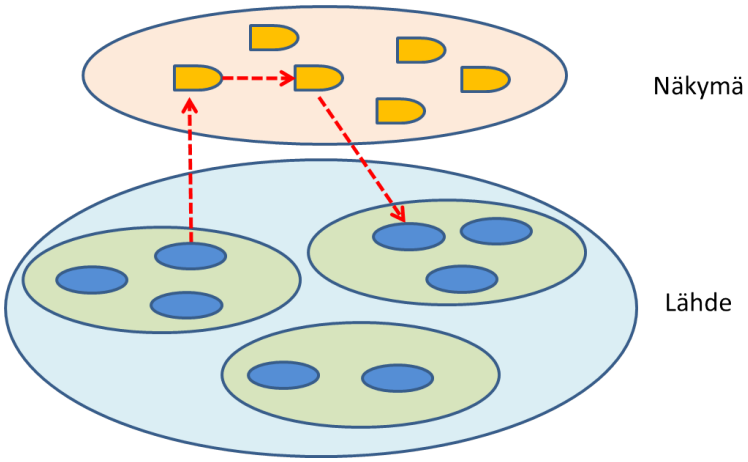
Linssit muodostavat selvän linssihierarkian (ks. kuvio 4) siten, että linsseille asetetaan edel-



Kuvio 7: Jos linssi toteuttaa PutGet- ja PutPut-lait, niin tällöin PutGet-laki määrää uudessa lähderakenteen olevan näkymän v', ja PutPut-laki määrää alkuperäisestä lähteestä saadun muuttumattomana pysyvän C:n pysymään muuttumattomana myös uudessa lähderakenteessa (mukaillen N. Foster 2010).

listen lakien lisäksi uusia lakeja. Tällöin siis useamman lain toteuttava linssi kuuluu luonnollisesti myös enemmän rajattujen linssien joukkoon. Mitä rajatumpia linssien ehdot ovat, sitä paremmin linssien käytös on ennustettavissa, mutta toisaalta liian suuret rajoitteet saattavat rajata pois kontekstin kannalta hyvinkin käyttökelpoisia linssejä (J. N. Foster 2010). Linssit voidaan karakterisoida käyttäytymisensä perusteella seuraavalla tavalla rajoitejärjestyksessä: kaikkien linssien joukko, linssit jotka toteuttavat PutGet-lain, hyvin käyttäytyvät linssit, erittäin hyvin käyttäytyvät linssit ja bijektiiviset linssit 4.

Edellämainittujen lakien lisäksi on olemassa myös muitakin lakeja, kuten esimerkiksi PutTwice-laki (J. N. Foster 2010). Tämä laki on lievennetty versio PutPut-laista ja kertoo sen, että jos sama päivitys tehdään kahdesti peräkkäin, niin lopputulos on sama kuin oltaisiin suoritettu vain yksi put-funktio kutsu. Kaikki hyvinkäyttäytyvät linssit toteuttavat PutTwice-lain (J. N. Foster 2010). Myös muita linssilakeja on kehitetty ja tutkittu (Fischer, Hu ja Pachero 2015), mutta tässä tekstissä käsitellään ainoastaan edellämainitut lait.



Kuvio 8: Jos linssi toteuttaa PutGet-lain, mutta ei PutPut-lakia, niin tällöin put-funktion käyttäytyminen on vähemmän rajoittunutta. Tällaisessa tilanteessa put-funktiokandidaatteja voi olla useita, ja oikean put-funktion valitsemiseen tarvitaan ylimääräistä informaatiota (mukaillen N. Foster 2010).

4 Linssit Haskelissa

Tässä kappaleessa käydään läpi hyvin yleisellä tasolla Twan van Laarhovenin linssit sellaisenaan, kuin ne olivat hyvin varhaisessa vaiheessa. Laarhovenin linssit ovat olleet pohjana Edvard Kmettin Haskell-ohjelmointikielen linssikirjastolle. Lisäksi tässä kappaleessa mainitaan yleisellä tasolla Edvard Kmettin linssikirjasto.

4.1 Twan van Laarhovenin linssit

Vuonna 2012 Edvard Kmett julkaisi ensimmäisen version linssikirjastosta, joka on toteutettu Haskelilla. Kyseenomainen kirjasto perustuu paljon alunperin Twan van Laarhovenin ideaan: funktionaalisiin viitteisiin (engl. functional references) (Laarhoven 2007). Seuraavaksi käydään hyvin pintapuolisesti läpi Haskelin linssien perusajatukset.

Laarhoven sanoo kirjoituksessaan funktionaalisen referenssin tarkoittavan tietorakennetta, jonka avulla voidaan muokata jonkun toisen rakenteen sisältöä. Myöhemmin Laarhoven ilmoitti käyttävänsä funktionaalisen referenssin sijasta termiä linssi. Laarhovenin esittämä alkuperäinen rakenne pitää sisällään get- ja set-funktiot, jotka ovat tyypiltään samanlaiset kuin linsseillä yleensäkin: get-funktiolla saadaan rakenteen sisällä oleva arvo, ja set-funktiolla asetetaan uusi arvo lähderakenteeseen. Seuraavaksi esitetään Laarhovenin varhainen määrittely Haskelin linssistä ja hänen esimerkkinsä linssistä, jolla voidaan muokata parin ensimmäistä arvoa (Laarhoven 2007)

```
data FRef s a = FRef { get :: s -> a , set :: a -> s -> s }
```

Ensimmäiseksi määritellään linssi. Data avainsanalla määritellään datatyyppi, jonka nimi on tässä FRef funktionaalisen referenssin mukaan nimettynä. FRef:n jälkeen tulevat s on lähderakenteen tyyppi ja a on lähderakenteen tyyppi. Tässä FRef on siis yleisellä tasolla määritelty linssi, jossa lähde- ja kohdetyypit on jätetty avoimeksi.

Yhtäsuuruus merkin jälkeen oleva FRef on linssin rakentaja, joka ottaa parametrinaan kaksi funktiota: get- ja set-funktiot. Laarhoven käyttää aiemmin mainitun put-funktion tilalla set-nimistä funktiota. Get-funktio ottaa ensimmäisenä argumenttinaan tyyppiä s olevan läh-

teen, ja palauttaa lähteestä tyyppiä a olevan näkymän. Set-funktio taas ottaa ensimmäisenä parametrina tyyppiä a olevan muokatun näkymän, toisen argumenttinä tyyppiä s olevan alkuperäisen lähteen ja palauttaa lähderakenteen kanssa samaa tyyppiä olevan muokatun lähteen. Seuraavaksi esitetään esimerkki Laarhovenin linssistä, jossa lähderakenteena on pari, jonka komponentit ovat tyyppiä x ja y, ja jolla voi muokata x:n arvoa.

```
fst :: FRef (x,y) x
fst =  FRef { get = \(x,y) -> x, set = \x' (x,y) -> (x',y) }
```

Laarhoven on määritellyt funktion fst, joka ei ota argumentteja, vaan palauttaa linssin, joka on tyyppiä Fref (x,y) x. Tässä linssin Fref lähdetyyppi s on pari (x,y) ja kohdetyyppi a on x. Nyt fst-funktiota kutsuttaessa se palauttaa linssin, jolla voidaan suorittaa transformaatio parille (x,y) ensimmäisen komponentin eli x:n suhteen. Toisin sanoen x voidaan saada ulos parista get-funktiolla, määritellä uusi x' ja laskea uusi pari set-funktiolla antamalla sille argumentteina x' ja (x,y):n, lopputuloksena (x',y).

Erityisen tärkeä huomio Laarhovenin linsseissä on se, että linssejä voi ketjuttaa funktiokompositioilla. Haskelissa funktiokompositio vastaa yhdistettyä funktiota matematiikassa. Tällä tavalla voidaan päästä Haskelissa muokkaamaan sisäkkäisiä rakenteita helposti, kuten esimerkiksi pareja, joiden komponentteinä on lisää pareja (Laarhoven 2007). Tämä on yksi yleinen käyttökohde linsseille Haskelissa, sillä Haskelissa on toisinaan työlästä muokata sisäkkäsiä rakenteita.

4.2 Edward Kmettin linssikirjasto

Suurta suosiota saanut, Edward Kmettin, Haskelin linssikirjaston ensimmäinen versio eli 0.1 ilmestyi vuonna 2012, ja tämän tekstin kirjoittamishetkellä kirjaston versionumero on 4.14. Linssikirjasto on liian laaja, jotta sitä voisi kuvailla tarkemmin tässä tutkielmassa. Edvard Kmett sanoo kirjaston linsseistä, että ne toteuttavat erittäin hyvin käyttäytyvien linssien lait (Kmett 2013a). Edward Kmett painottaa kuitenkin sitä, että ohjelmoija ei voi luottaa siihen, että tyyppijärjestelmä takaa lakien pysyvyyden, vaan ohjelmoijan on itse varmistettava GetPut-, PutGet- ja PutPut-lakien voimassaolo (Kmett 2013b). Haskelissa on myös muitakin linssikirjastoja kuten esimerkiksi fclabels, data-lens ja yall, mutta ne eivät ole saavuttaneet

niin suurta suosiota kuin Edward Kmettin nykyinen linssikirjasto.

5 Yhteenveto

Tässä tutkielmassa perehdyttiin datan transformointiin ohjelmoinnin näkökulmasta. Yleinen tapa toteuttaa datan transformointi datamallista toiseen on toteuttaa transformaatiot yksisuuntaisilla ohjelmilla, mutta tämä tapa saattaa aiheuttaa vaikeuksia ohjelmien toteuttamisessa, ymmärtämisessä ja ylläpidossa. Tässä tekstissä esiteltiin toisenlainen lähestymistapa kahden eri mallin väliseen transformaatioon, nimittäin kaksisuuntaisiin transformaatiot.

Kaksisuuntainen transformaatio on ohjelma, joka koostuu kahdesta yksisuuntaisesta transformaatiosta, ja jonka avulla datamallien väliset transformaatiot voidaan toteuttaa. Kaksisuuntaisten tranformaatioiden yhteydessä käytiin läpi myös niin sanottu näkymänpäivitysongelma, joka yleistettynä liittyy vahvasti datamallien välisiin transformaatioihin. Erityisen tarkasti käytiin läpi kaksisuuntaiset transformaatiot nimeltään linssit. Linssit esiteltiin melko yleisellä tasolla, mutta tekstissä pyrittiin kuitenkin käymään läpi hiukan yksityiskohtaisemmin linsseihin liittyvät niin sanottujen get- ja put-funktioiden vaatimukset ja rajoitteet. Lopuksi tekstissä esitettiin lyhyesti Haskell-ohjelmointikielen Edward Kmettin linssikirjaston perusteita, jotka liittyvät vahvasti niin sanottuihin van-Laarhovenin linsseihin.

Käytännössä, vaikka linssien teoriaa on kehitetty tiedeyhteisöissä paljon, vaikuttaa siltä, ettei linssit ole kovinkaan yleisesti käytössä. Linsseihin perustuvat ohjelmointikielet ovat vielä tänä päivänä melko harvinaisia. Esimerkiksi Boomerang on funktio-ohjelmointikieli, joka perustuu nimenomaan linsseihin. Erityistä huomiota herätti C-kielellä toteutettu Augeas-kirjasto, joka käyttää linssejä toteutuksessaan. Linssit vaativat käytännössä hyvin rikkaan tyyppijärjestelmän toimiakseen, joten olisi ollut mielenkiintoista perehtyä siihen, miten C-kielellä kirjoitettu Augeas kirjasto on toteutettu. Lisäksi Haskelin linssit, ja Haskelin linssikirjastoon uutena tulleet prismat olisivat hyvin kiinnostava tutkimuskohde.

Muita hyviä tutkimuskohteita olisi yksi- ja kaksisuuntaisten transformaatioiden esittely ja vertailu, ja se, voitaisiinko kaksisuuntaisia transformaatioita tai linssejä toteuttaa esimerkiksi olio-ohjelmoinnilla. Tässä tekstissä kaksisuuntaisiin transformaatioihin keskityttiin lähinnä funktio-ohjelmoinnin näkökulmasta.

Lähteet

Aaron Bohannon, Benjamin C. Pierce, ja Jeffrey A. Vaughan. 2006. “Relational lenses: a language for updatable views”. Teoksessa *PODS 2006: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* 338–347. ACM New York.

Bancilhon, F., ja N. Spyratos. 1981. “Update Semantics of Relational Views”. Teoksessa *ACM Transactions on Database Systems,* 557–575. ACM New York.

Bohannon, Aaron, Benjamin C. Pierce ja Jeffrey A. Vaughan. 2006. “Relational lenses: A Language for Updatable Views”. Teoksessa *Principles of Database Systems (PODS),* 338–347. ACM New York.

Czarnecki, Krzysztof, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmer, Andy Schurr ja James F. Terwilliger. 2009. “Bidirectional Transformations: A Cross-Discipline Perspective: GRACE meeting notes, state of art, and outlook”. Teoksessa *Theory and Practice of Model Transformations,* 260–283. Springer.

Fischer, Sebastian, Zhenjiang Hu ja Hugo Pachero. 2015. “A Clear Picture of Lens Laws”. Teoksessa *Mathematics of Program Construction,* 215–223. Springer.

Foster, J. N., M. B. Greenwald, J. T. Moore, B. C. Pierce ja A. Schmitt. 2007. “Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem”. Teoksessa *ACM Transactions on Programming Languages and Systems,* nide 29(3):17. Springer.

Foster, John Nathan. 2010. “Bidirectional Programming Languages”. Tohtorinväitöskirja, Univercity of Pennsylvania.

Foster, Nate. 2010. *Bidirectional Programming*. Saatavilla `http://www.cs.cornell.edu/~jnfoster/papers/ssgip/jnfoster-ssgip1.pdf`.

J. Nathan Foster, Benjamin C. Pierce, ja Steve Zdancewic. 2009. “Updatable Security Views”. Teoksessa *2009 22nd IEEE Computer Security Foundations Symposium,* 60–74. IEEE.

Kmett, Edward. 2012. *Haskell linssikirjasto: https://github.com/ekmett/lens.*

———. 2013a. *https://github.com/ekmett/lens/wiki/FAQ.* Saatavilla githubissa 2016.

———. 2013b. *https://github.com/ekmett/lens/wiki/Overview.* Saatavilla githubissa 2016.

Laarhoven, Twan van. 2007. *twanvl.nl/blog/haskell/overloading-functional-references.*

Lipovaca, Miran. 2011. *Learn You a Haskell for Great Good!: A Beginner's Guide.* No Starch Press. ISBN: ISBN 10: 1593272839 / ISBN 13: 9781593272838.

Lutterkort, D. 2007. *A Linux configuration API: http://augeas.net.*

Pierce, Benjamin C. 2012. "Linguistic Foundations for Bidirectional Transformations: Invited Tutorial". Teoksessa *Principles of Database Systems (PODS),* 61–64. ACM New York.

Stevens, Perdita. 2007. "A Landscape of Bidirectional Model Transformations". Teoksessa *Generative and Transformational Techniques in Software Engineering 2,* 408–424. Springer.

———. 2008. "Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions". Teoksessa *Software and System Modeling,* 7–20. Springer-Verlag.

Wagner, Daniel. 2014. "Symmetric Edit Lenses: A New Foundation for Bidirectional Languages". Tohtorinväitöskirja, Univercity of Pennsylvania.