Hyväksytty kieleltään

# Haskellin erityispiirteitä

Tuomas Starck

Kypsyysnäyte
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 28. huhtikuuta 2017

Lähes koko kuluneen tietotekniikan historian ajan on imperatiivinen strukturoitu ohjelmointi ollut vallitseva paradigma ohjelmistoja kehitettäessä. Imperatiivisten kielten, kuten esimerkiksi C:n tai Javan, suosio on niin merkkitävä, että monelle ohjelmoijalle vaihtoehdot ovat jääneet suurelta osin tuntemattomiksi. Haskell on funktionaalinen, puhdas ja löyhä ohjelmointikieli, jonka poikkeukselliset ominaisuudet erottavat sen paitsi kaikista imperatiivisista kielistä niin myös monista muista funktionaalisista kielistä.

Funktionaalinen ohjelmointi on ohjelmointiparadigma, jossa ohjelma koostuu kuvauslauseista (engl. *declarations*) eikä käskylausekkeista (engl. *statement*) niin kuin imperatiivisessa ohjelmoinnissa. Funktionaalisen ohjelmoinnin juuret ovat Alonzo Churchin lambdakalkyylissä. Sekä funktionaalisessa ohjelmoinnissa että lambdakalkyylissä laskenta tarkoittaa funktioiden sieventämistä, mistä johtuen Haskellin funktiot ovat paitsi nimeltään niin myös merkitykseltään vastaavia matemaattisten funktioiden kanssa – molemmat palauttavat samalla syötteellä aina saman arvon.

Haskellia kutsutaan usein laiskan evaluaation kieleksi, mutta tarkalleen ottaen Haskell-kielen määrittelyssä sanotaan, että sievennysstrategia on löyhä (engl. *non-strict*). Löyhyys tarkoittaa sitä, että funktioita sievennetään ulkoa sisälle. Esimerkiksi lausekkeesta `a + (b * c)` lasketaan ensin `+`-operaatio. Löyhyys mahdollistaa muun muassa äärettömien tietorakenteiden määrittelyn ja käsittelyn, sillä Haskellin suoritusympäristö ei pyri evaluoimaan niitä loppuun saakka, joten ohjelma ei jää niiden kohdalla jumiin.

Käytännössä löyhyys Haskellissa on toteutettu laiskuuden avulla. Laiska evaluaatio tarkoittaa sitä, että ohjelman lausekkeet evaluoidaan vasta silloin, kun niiden palauttamaa arvoa tarvitaan laskennan jatkamiseksi. Aiemmassa esimerkissä `a + (b * c)` laiskuus tarkoittaa sitä, että Haskellin suoritusympäristö pyrkii ensin laskemaan `+`-operaation kuten löyhyys määrittelee. Koska `+`-operaatio tarvitsee `(b * c)` -laskun arvon, muodostuu riippuvuus, joka aiheuttaa sulkeiden sisäisen kertolaskun evaluoinnin.

Haskellin luonnin aikaan laiskuudesta oli jo noin vuosikymmenen verran kokemusta ohjelmointikieliä kehitettäessä, ja tiedettiin, ettei laiskuus ei ole täysin ongelmaton valinta. Laiskan suorituksen on pidettävä kirjaa siitä, mitkä osat ohjelmaa ovat suorittamatta, mikä kasvattaa yleiskustannuksia. Kirjanpidosta seuraava hidastus on kuitenkin vakioinen ja verrattain pieni. Suurempi ongelma on se, että laiskan suorituksen tilavaativuutta on vaikea arvioida ja se saattaa kasvaa ylilineaarisesti. Haskell ei siksi ole täysin laiska kieli, sillä siihen on lisätty sekä keinoja pakottaa lausekkeen ahne evaluaatio, että tietorakenteita, jotka eivät ole laiskoja. Tällöin ohjelmoija voi poistaa laiskuutta sellaisissa ohjelman kohdissa, joissa siitä olisi haittaa.

Puhdas funktionaalinen ohjelmointi tarkoittaa sitä, että funktioiden toiminta pelkistyy vain ja ainoastaan parametrien vastaanottamiseen ja arvon palauttamiseen – sivuvaikutukset eivät ole sallittuja. Sivuvaikutuksilla tarkoitetaan ulkoista tilanmuutosta, joka voi olla esimerkiksi muistinkäsittelyä

tai siirräntää (engl. *input/output*).

Puhtaus liitetään usein erottamattomasti funktionaalisuuteen, mutta epäpuhtaita funktionaalisia kieliä on olemassa huomattavasti enemmän kuin puhtaita. Koska Haskell on puhdas ei siinä ole muuttujia, ei sijoitusoperaatiota, ei funktioiden ulkopuolista tilaa, ei muistinkäsittelyä, eikä vapaata siirräntää. Puhtaus on luonnollinen seuraus laiskuudesta. Koska laiskuuden takia evaluointijärjestys määräytyy tarpeen mukaan, on siirrännän tai muiden tilamanipulaatioiden mielekäs toteuttaminen vaikeaa.

Koska siirräntä on kuitenkin oleellinen ominaisuus mille tahansa vakavasti otettavalle ohjelmointikielelle, on Haskellin kehityksessä nähty paljon vaivaa sen mahdollistamiseksi. Ratkaisu, johon on päädytty, on monadinen siirräntä, mikä tarkoittaa sitä, että siirräntää voi tehdä vain `IO`-monadissa. `IO`-monadi on tyyppiluokkien avulla toteutettu korkeamman asteen tyyppi, joka ikään kuin kapseloi siirrännän omaan kontekstiinsa. Tällöin kielen puhtaus ja tyyppiturvallisuus säilyy, mutta siirräntä on mahdollista.

Puhtauden ja löyhyyden lisäksi Haskellissa on monia ominaisuuksia, jotka ainakin sellaisenaan ovat verrattain harvinaisia muissa ohjelmointikielissä. Toisaalta ohjelmointikieliä on lukuisia, joten on hankala mainita mitään yksittäistä Haskellin ominaisuutta, jota muista kielistä ei löytyisi. Esimerkiksi staattisen tyypityksen ja tyyppipäättelyn yhdistelmä ei ole aivan arkinen ominaisuus, mutta Haskellin lisäksi se löytyy Scalasta. Tyyppiluokat ovat Haskellille erityiset sellaisena kuin ne on Haskellissa toteutettu, mutta hyvin samankaltaisia ominaisuuksia löytyy monista muista kielistä, vaikka niillä olisi eri nimi. Monadit tunnetaan yleisesti Haskellin erityispiirteenä, mutta ne eivät oikeastaan ole kielen vaan standardikirjaston ominaisuus, jonka voisi toteuttaa missä tahansa kielessä.

Lopulta juuri puhtauden ja löyhyyden yhdistelmä on se erityispiirre, joka on ainutlaatuinen Haskellille. Joskaan ei tietysti täysin ainutlaatuinen, sillä samat ominaisuudet ovat Mirandassa ja Idrisissä. Miranda oli Haskellia edeltävä kieli, josta Haskell otti paljon vaikutteita. Idris sen sijaan on hyvin uusi kieli, joka on rakennettu suoraan Haskellin päälle ja siksi se perii suuren joukon Haskellin ominaisuuksia. Ohjelmointikielten kehitys ei tapahdu tyhjiössä.