TIK.kand tutkimussuunnitelma:

Laiska evaluointi funktionaalisessa ohjelmoinnissa

Atte Keinänen
Aalto-yliopisto
atte-keinanen@aalto.fi

7. helmikuuta 2016

Kandidaatintyön nimi: Laiska evaluointi funktionaalisessa ohjelmoinnissa

Työn tekijä: Atte Keinänen, Informaatioverkostot

Ohjaaja: Juha Sorva, Tietotekniikan laitos

1 Tiivistelmä tutkimuksesta

Ohjelmointikielissä on vaihtelevia evaluointistrategioita, eli käytäntöjä sille, milloin ja miten lausekkeen arvo evaluoidaan. Valtavirtaa ohjelmoinnissa on ahne evaluointi, eli lauseke evaluoidaan välittömästi, kun lauseke sijoitetaan muuttujaan.

Kuitenkin suosioon on noussut Haskellin, OCamlin ja Scalan kaltaisia ohjelmointikieliä, jotka tukevat myös laiskaa evaluointia. Tällä tarkoitetaan sitä, että lausekkeen arvo evaluoidaan vasta, kun sen arvoa tarvitaan. Laiskalla evaluoinnilla pystytään välttämään tarpeettomia lausekkeiden evaluointeja, ja toisaalta se mahdollista päättymättömän listan kaltaisia tietorakenteita. Toisaalta laiskaa evaluointia kritisoidaan siitä, että sitä käyttävien ohjelmien muistinkulutusta ja koodin suoritusjärjestystä on hankalaa ennakoida.

Kandityössä tarkastellaan laiskan evaluoinnin merkitystä nykypäivänä, tarkastellen sekä historiaa, leviämistä eri ohjelmointikieliin että sovellutuksia työelämässä ja tutkimuksessa. Lisäksi evaluoinnin hyötyjä ja haittoja tarkastellaan erikseen erilaisissa laiskan evaluoinnin sovellutuksissa.

2 Tavoitteet ja näkökulmat

Halusin valita itse kandiaiheeni, koska puhdas funktionaalinen ja erityisesti Haskell kiinnostivat minua. En ollut ennestään käyttänyt puhtaasti funktionaalisia ohjelmointikieliä, ja olin työpaikkani kautta kuullut, että Haskelliin tutustuminen olisi hyvä tapa oppia ymmärtämään funktionaalista ohjelmointia ja sen kehitystä syvemmin. Ennen aiheen valintaa luin kirjan Learn You a Haskell for Great Good [2] ja Haskellin tyyppijärjestelmää

käsittelevän Typeclassopedian [3]. Nämä auttoivat löytämään kiinnostavia aihepiirejä Haskellin liittyen.

Päädyin rajaamaan aiheen laiskaan evaluointiin. Laiska evaluointi on monille puhtaasti funktionaalisille ohjelmointikielille yhteistä. Tutkimuskysymyksinäni ovat (1) laiskan evaluoinnin merkitys nykypäivänä sekä (2) laiskan evaluoinnin hyödyt ja haitat. Tavoitteena on, että työn tuloksista voisi olla myös aihepiirin yliopisto-opetusta suunnitellessa iloa.

3 Tutkimusmateriaali

Laiskasta evaluoinnista on runsaasti akateemisia artikkeleita. Esimerkiksi Haskell -ohjelmointikielen kehityksen taustalla oli useita jo klassikonomaiseen asemaan nousseita artikkeleita, joissa perusteltiin laiskan evaluoinnin hyötyjä [1]. Samoin erilaisista spesifeistä ongelmista esimerkiksi tekoälytutkimuksen saralta, joiden käytössä laiska evaluointi on osoittautunut hyödylliseksi, on paljon saatavilla.

Akateemisten artikkeleiden lisäksi luontevaa on seurata funktionaalisen ohjelmoinnin piirissä arvostettujen ihmisten blogeja ja kirjoituksia. Näiden avulla saadaan vastattua pelkkiä akateemisia tutkimuksia kattavammin siihen, mikä on käsitys laiskan evaluoinnin suosiosta ja käyttökohteista tällä hetkellä.

Aikaa yksittäisen artikkelin menee luultavasti noin 30 minuuttia per artikkeli. Koska kyseessä on melko tyylipuhdas kirjallisuustutkimus, artikkeleita tarvitsee kerätä arviolta 20-30. Lukemisen lisäksi aikaa menee kerätyn tiedon järjestämiseen, mistä lisää seuraavassa luvussa.

4 Tutkimusmenetelmät

Kerätyt artikkelit kootaan Trello-järjestelmään aihepiireittäin siten, että kullekin artikkelille luodaan oma Trello-korttinsa, josta on linkki alkuperäiseen artikkeliin. Yksittäinen artikkeli voi kuulua useampaan aihepiiriin. Aihepiirijako on alkuvaiheessa seuraavanlainen, ja sitä muutetaan ja kasvatetaan lähdemateriaalin määrän kasvaessa:

- Varhaishistoria
- Vaikutukset tietojenkäsittelytieteeseen
- Toteutus nykyaikaisissa ohjelmointikielissä
- Akateeminen käyttö tänä päivänä
- Yrityskäyttö tänä päivänä

Kustakin artikkelista eritellään Trellossa kommenttitoiminnon avulla poimintoja ja muistiinpanoja aihepiireittäin. Todennäköisesti päädyn käyttämään yhdistelmää suoria lainauksia artikkeleista että itse kirjoittamiani lyhyitä tiivistelmiä.

Huomionarvoista on myös, että aina kun lisään artikkelin Trelloon, lisään sen myös BibTeX-lähdeluetteloon. Näin varsinaista kandidaatintyötä kirjoittaessani minun ei tarvitse enää erikseen etsiä lähdeviittauksia.

Aineiston keruun Trello-taulu on avoimesti nähtävillä seuraavassa osoitteessa:

`https://trello.com/b/Q9ZenGG4`

Versioin kandikurssin LaTeX-materiaalit GitHubissa. GitHub-repositoryni on Trello-taulun tapaan julksisesti nähtävillä, ja sijaitsee seuraavassa osoitteessa:

`https://github.com/attrck/kandi`.

Artikkelien keruun lisäksi kerään mielipiteitä laiskan evaluoinnin menetelmistä artikkeleiden lisäksi blogeista tuttavapiiristäni. Olen kartoittanut tämän suhtautumista itse raportin sisältöön tarkemmin osassa 8.

5 Haasteet

Työn haasteet liittyvät toisiinsa liittyen työn rajaukseen ja siihen, että materiaalia on huomattavan paljon saatavilla.

Laiskan evaluaation sisältä pystyisi kandin melko suppean laajuuden huomioiden käsittelemään halutessaan vain yksittäistäkin osa-aluetta, esimerkiksi sen hyödyntämistä yritysmaailmassa, tai teknistä toteutusta parissa eri ohjelmointikielessä. Tämä on hyvä pitää mielessä, jos vaikuttaa siltä, jos kandin sivumäärä meinaa karata käsistä.

Tehdyn tutkimuksen suuri määrä johtaa ennen kaikkea siihen, että tarjolla olevaa artikkelivalikoimaa on hyvä arvioida ennen artikkeleihin syventymistä, ja näin valita tieteellisesti vakuuttavimmat ja oman aiheen kannalta relevanteimmat artikkelit. Trelloa voisi luontevasti käyttää siihen, että ensiksi koostaa laajemman listan artikkeleista, jota tarvittaessa karsii, ja sitten vasta lukee artikkelit syvällisemmin ja tekee niistä muistiinpanot.

6 Resurssit

Työssä ei ole muita osapuolia kuin minä ja työn ohjaaja Juha Sorva. Juhan kanssa olemme tähän asti saaneet hyvin sujuvasti sovittua yksittäiset tapaamiset, ja sen kanssa tuskin tulee jatkossakaan ongelmia. Tapaamme hieman kurssin virallista tapaamissykliä tiheämmin, koska suoritan kurssin tavallisesta poikkealla aikataululla.

7 Aikataulu

Kandityöstä on tavoitteenani saada V3, eli viimeistelemätön mutta täyspitkä versio, valmiiksi jo 24. helmikuuta mennessä. Syynä on, että lähden 25. helmikuuta vaihto-opiskelemaan Etelä-Koreaan, enkä halua jättää työtä täysin keskeneräiseksi siinä kohtaa. Tapaan kandiohjaajaani 23. helmikuuta.

Yhdistän kandin suorituksen kesäkandikurssiin. Palattuani Etelä-Koreasta heinäkuun lopussa jatkan kandityön viimeistelyn parissa, ja osallistun elokuussa niille luennoille, joihin en nyt keväällä ehtinyt. Hoidan seminaarin, opponoinnin ja kypsyysnäytteen kesäkurssin aikataulusta riippuen elokuun lopulla tai syyskuun alussa.

8 Raportin rakenne

Haluan kandista tarinamaisen, kielellisesti monipuolisen ja innostavan lukea, joten rakenteen suunnittelu noudattaa seuraavia periaatteita:

- Rakennetta mietitään kronologisesti, tavoitteet edellä: miten osio asettuu osaksi tarinan eteenpäin kehittyvää jatkumoa, miten osio ylläpitää lukijan mielenkiintoa?
- Jaan tekstin lukuihin ja alaotsikoihin vasta siinä kohtaa, kun tekstimassaa on jo enemmän. Pyrin näin estämään sen, että liian tiukka jaottelu rajoittaisi kielellistä ilmaisua tai tarinan loogista etenemistä.

Suunniteltu rakenne:

1. Aloitan tarinan todellisen elämän esimerkillä, jossa on tullut vastaan aito tarve sille, että koodia ei evaluoida tiukasti. Eläväisintä olisi saada oikein kouriintuntuva esimerkki tilanteesta, jossa ollaan joduttu ongelmiin, kun laiskaa evaluointia ei ole aluksi käytetty. Esimerkillä pyrin herättämään lukijan mielenkiinnon, ja nivomaan aiheen välittömästi konkretiaan ennen siihen syventymistä.
2. Kerron, kuinka 2000-luvulla on näkynyt paljon kehitystä, jossa ajatusta siitä, että lausekkeita ei ole aina pakko evaluoida välittömästi kun ne koodissa määritellään, on sovellettu laajemmalle. Esimerkiksi uudet ohjelmointikielet kuten Scala tukevat sitä, ja monet laiskan evaluoinnin ajatusta hyödyntävät kirjastot ovat nähneet päivänvalon esimerkiksi JavaScriptissä. Jos tuntuu luontevalta, niin samassa yhteydessä voi antaa lukijalle alustavan intuition laiskasta evaluoinnista vertauskuvia hyödyntäen, kuitenkin korostaen että kyseessä analogia, ei eksakti käsitemäärittely.
3. Määrittelen sitä tietotekniikan kontekstia, jossa nyt liikutaan, ja määrittelen keskeisiä käsitteitä ylätason käsitteistä yksityiskohtaisempiin. Ensiksi kerron mistä koodin evaluoinnissa on kyse, sitten mitä evaluointistrategia tarkoittaa ja millaisia erilaisia evaluointistrategioita on olemassa (vielä kuitenkaan vertailematta niiden vahvuuksia/heikkouksia). Lopulta tarkennan laiskaan evaluointiin sekä siihen tiiviisti liittyviin käsitteisiin (mm. call-by-need, call-by-name ja thunk). Lukija saa tästä tarvittavaa sanavarastoa tekstin lukemiseen.
4. Kerron, että olen hakemassa vastausta kahteen tutkimuskysymykseen: mikä sen merkitys tietotekniikassa on nykyhetkellä, ja millaisia vahvuuksia tai heikkouksia laiskalla evaluoinnilla on. Samassa yhteydessä kerron perustellen, että olen tarkoituksella jättänyt laiskan evaluoinnin matemaattisen ja kääntäjäteknisen tarkastelun tämän työn ulkopuolelle. Myös työn muista tavoitteista mainitseminen on mahdollista: voin vaikka mainita siitä, että toivon työn tulosten olevan hyödynnettävissä ohjelmoinnin opetuksessa.
5. Kerron käyttämäni tieteelliset menetelmät (kirjallisuuskatsaus ja mielipidekartoitus) ja perustelen miksi päädyin juuri näihin menetelmiin. Valotan kirjallisuuskatsauksen prosessia: mitä hakusanoja käytin, mitä tietokantoja hyödynsin ja millaisia määriä artikkeleita kävin läpi. Käyn samaan tapaan läpi mielipidekartoituksen toteutuksen, eli sekä tunnettujen kirjoittajien blogien läpikäynnin että kokemusten keräämisen tuttavapiiristäni.

6. Esittelen työn rakennetta perustellen sen menetelmien ja tutkimuskysymysten avulla. Kerron, että käsittelen ensin katsausmaisesti laiskan evaluoinnin varhaista historiaa, minkä jälkeen poraudun ensimmäiseen tutkimuskysymykseen laiskasta evaluoinnista nykypäivänä. Sitten, kun lukijalla on hyvä yleiskuva nykypäivän sovellutuksista, on sujuvaa siirtyä puhumaan yksittäisten sovellutusten yksityiskohdista sekä vahvuuksista ja heikkouksista.

7. Nykypäivän tilannetta taustoitan aluksi kertomalla, kuinka idea laiskasta evaluoinnista on saanut alkunsa akateemisena ideana, jota alettiin kokeilla erilaisissa ohjelmointikielissä, ja jopa prosessoriarkkitehtuureissa. Tästä siirryn Haskellin kehitykseen, ja siihen, kuinka laiskan evaluoinnin vaatimus osaltaan johti puhtaasti funktionaalisen ohjelmoinnin paradigman kehittymiseen. Käsittelyssä tässä kappaleessa siis käytännössä noin vuodet 1975-2000. Tässä elävöitän tekstiä historia-anekdooteilla.

8. Käyn läpi sitä, kuinka Haskell ja laiska evaluointi alkoi inspiroimaan monia uusia ohjelmointikieliä ja ohjelmointikielien kirjastoja. Tässä kohtaa keskityn nimenomaan siihen diversiteettiin, eli millaisissa erilaisissa muodoissa laiska evaluointi nykyään esiintyy, ja kuinka sitä käytetään myös tiukan evaluoinnin rinnalla. Käyn läpi paljon esimerkkejä laiskan evaluoinnin sovellutuksista. Jos esimerkkien määrä on suuri, se voi aiheuttaa haasteita luvun jäsentämiselle.

9. Nyt valitsen kiinnostavimmat aihealueet/sovellutukset tarkempaan tarkasteluun, ja otan tässä kohtaa myös vahvuus/heikkous -selvityksen mukaan. Perustelen, että laiskaa evaluointia ei ole mieltä tutkia yhtenä kokonaisuutena mielipidearvioinnin keinoin, vaan on järkevämpää keskittyä yksittäisiin sovellutuksiin. Aiheita/sovellutuksia voisi olla esimerkiksi Haskell, Scala, JavaScript-kirjastot ja FRP. Tässä voi saada aikaan mielenkiintoista mielipiteiden, blogien ja tieteellisten artikkeleiden vuoropuhelua.

10. Mielipiteistä ja sovellutuksista voisi jatkaa vielä kertomalla laiskan evaluoinnin hyödyntämisestä tieteellisessä tutkimuksessa, kuten laskennallisessa biologiassa. Myös yritysesimerkkejä voi ottaa mukaan. Tämä olisi teoriassa yhdistettävissä edelliseen lukuun, mutta toisaalta koen tämän olevan kuitenkin korkeammalla tasolla (kielten ja kirjastojen vahvuudet/heikkoudet vs. laiskan evaluoinnin soveltuvuus yleisemmin eri alojen työhön).

11. Tarinan kaaren kannalta olisi kandin loppupuolella kiva olla tulevaisuuden tarkastelua. Tässä voisin nostaa vielä pari aikaisemmin raportissa käsiteltyä trendiä, jotka vaikuttavat lupaavilta tulevaisuuden osalta, erikseen tarkasteluun. Voin myös koota yhteen, millaisten esteiden raivaaminen voisi auttaa laiskaa evaluointia entistä suurempaan suosioon.

12. Yhteenvedon yhteydessä arvioidaan vielä työn metodisia heikkouksia/rajoituksia, ja mitä puutteita niistä on mahdollisesti seurannut. Sitä, mitä muuta yhteenvedossa olisi mielekästä käsitellä, pitää vielä selvittää.

Lähteet

[1] Paul Hudak, John Hughes, Simon Peyton Jones, ja Philip Wadler. A history of haskell: being lazy with class. Kirjassa *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, ss. 12–1. ACM, 2007.

[2] Miran Lipovaca. *Learn you a haskell for great good!: a beginner's guide*. no starch press, 2011.

[3] Brent Yorgey. The typeclassopedia. *The Monad. Reader Issue 13*, s. 17, 2009.