Induktiivinen logiikkaohjelmointi

Marko Oinonen

Helsinki 13.4.2004

Ohjelmointikielten periaatteet
Seminaarityö

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Induktiivinen logiikkaohjelmointi on koneoppimisen menetelmä, joka kehittyi tarpeesta laajentaa perinteisten propositionaalisten menetelmien ilmaisuvoimaa predikaattilogiikan suuntaan. Induktiivisessa logiikkaohjelmoinnissa tavoitteena on esimerkkien ja rajoitusten avulla automaattisesti induktiivista päättelyä käyttäen tuottaa logiikkaohjelmia [LaD93].

Logiikkaohjelmointi on 1970-luvulla lähinnä Prolog-ohjelmointikielen kautta alkunsa saanut deklaratiivinen ohjelmointiparadigma, jossa ohjelmat kuvataan predikaattilogiikan lauseina. Lauseet muodostavat tietokannan, jonka avulla päätellään vastauksia käyttäjän esittämiin kyselyihin [ClM95].

Deklaratiivisen luoteensa takia logiikkaohjelmat sopivat hyvin kuvaustavaksi ilmiöiden mallintamiseen. Juuri tästä syystä Prolog on valittu esitystavaksi induktiivisessa logiikkaohjelmoinnissa. Induktiivista logiikkaohjelmointia käytetään muodostamaan esimerkkien pohjalta yleisiä malleja, joita voidaan käyttää joidenkin vaikeiden käytännön ongelmien ratkaisemiseen. Olemassa olevista induktiivisista logiikkaohjelmointijärjestelmistä on dokumentoitu onnistuneita kokeiluja sairauksien tunnistamisesta oireiden perusteella aina proteiinin kolmiuloitteisen rakenteen ennustamiseen aminohapporakenteen avulla [MuD94].

Puhtaasti ohjelmoinnin näkökulmasta induktiivinen logiikkaohjelmointi voidaan nähdä yhtenä muotona induktiivisesta ohjelmoinnista, jossa tavoitteena on ajettavien ohjelmien automaattinen synteesi esimerkkien pohjalta. Näin ajateltuna induktiivinen logiikkaohjelmointijärjestelmä voidaan nähdä kääntäjänä ja annetut esimerkit sekä asetut rajoitteet induktiivisena logiikkaohjelmana, josta kääntäjä tuottaa Prolog-ohjelman [BeG95].

Induktiivinen ohjelmointi tuo mukanaan uudenlaisen tavan vertailla ohjelmointiparadigmoja. Imperatiivinen-deklaratiivinen -akselin lisäksi voidaan tarkastella paradigmoja deduktiivinen-induktiivinen -akselilla. Deduktiivisessa ohjelmoinnissa yleisellä tasolla esitettyjen vaatimusten pohjalta tuotetaan käsityönä ohjelma. Induktiivisessa ohjelmoinnissa ohjelma tuotetaan automaattisesti ohjelman haluttua toimintaa kuvaavien esimerkkien pohjalta [Par97]. Esimerkkien ja mahdollisten rajoitusten esittämiseen käytettyä kieltä kutsutaan induktiiviseksi logiikkaohjelmointikieleksi [BeG95].

2 Logiikkaohjelmointi

Logiikkaohjelmointi on deklaratiivinen ohjelmointiparadigma. Logiikkaohjelma koostuu joukosta predikaattilogiikan lauseita, joiden avulla etsitään vastauksia käyttäjän tekemiin kyselyihin. Logiikkaohjelmointikieliä on useita, mutta käytännön merkitystä niistä on vain Prolog-kielellä. Siksi tästä eteenpäin logiikkaohjelmoinnista ja Prologista puhutaan toistensa synonyymeinä [StS86].

Logiikkaohjelmoinnissa ei hyödynnetä koko predikaattilogiikan ilmaisuvoimaa, vaan ohjelmissa käytettävät lauseet on rajattu Hornin lauseisiin. Nämä esitetään muodossa B :- $A_1, \ldots A_n$, missä B ja A_i ovat *literaaleja* eli predikaatista ja sen attribuuteista koostuvia *atomeja* tai atomien negaatioita. Kaksoispiste ja viiva kuvaavat implikaatiota, jonka suunta on oikealta vasemmalle. Pilkku kahden literaalin välissä tarkoittaa konjunktiota. Edellä mainittu sääntö tulkitaan siis "jos A_1 ja $\ldots$ja A_n, niin B". Mikäli $n = 0$ kutsutaan lausetta *faktaksi*. Muutoin lausetta kutsutaan *säännöksi*. Säännön implikaation vasenta puolta eli johtopäätöstä kutsutaan lauseen *pääksi*. Implikaation oikeaa puolta eli lauseen ehtoa kutsutaan lauseen *vartaloksi*. Joukko sääntöjä, joiden päänä on sama predikaatti, muodostavat kyseisen *predikaatin määritelmän* [StS86].

Prolog-ohjelmaa käytetään yleensä suorittamalla kyselyjä ohjelman muodostamaan tietokantaan. Kyselyt esitetään muodossa ?- $A_1, \ldots, A_n$. Prolog-tulkki yrittää yksi kerrallaan toteuttaa lauseiden $A_1, \ldots, A_n$ sisältämät tavoitteet [StS86].

3 Induktiivinen ohjelmointi

Deduktiivinen päättely tarkoittaa päättelyä yleisestä erityiseen. Induktiivinen päättely taas tarkoittaa päättelyä erityisestä yleiseen. Logiikkaohjelmissa vastauksen etsiminen kyselyyn perustuu deduktiiviseen päättelyyn. Esimerkiksi Prolog-ohjelmasta

```
kuolevainen(X) :- ihminen(X).
ihminen(sokrates).
```

saadaan kyselylle

```
?- kuolevainen(sokrates).
```

myöntävä vastaus. Tässä yleistä sääntöä käyttäen pääteltiin yksittäistä esimerkkiä koskeva tieto. Deduktiivinen päättely on täsmällistä ja sen tulos pitää aina paikkansa [BeG95, Par97].

Induktiivisella päättelyllä faktoista

```
ihminen(sokrates).
kuolevainen(sokrates).
```

voitaisiin tehdä yleistys

```
kuolevainen(X) :- ihminen(X).
```

Induktiivinen päättely ei ole täysin pitävää, koska lähtökohtana on joukko yksittäisi esimerkkejä. Päättelyn lopputulos on näiden esimerkkien perusteella tehty yleistys [Par97].

Perinteisesti ohjelmoinnin tavoitteena on tuottaa ajettava ohjelma lähtökohtana vaatimukset, jotka kuvaavat ohjelmointikieltä korkeammalla abstraktiotasolla ohjelmalta vaadittavan toiminnallisuuden. Tämän prosessin voidaan ajatella olevan deduktiivista päättelyä eli päättelyä yleisestä erityiseen. Ongelmana usein on vaatimusten epätäydellisyys. Prosessi perustuu kuitenkin oletukseen vaatimusten täydellisyydestä [Par97].

Induktiivisen ohjelmoinnin ajatuksena on hyödyntää induktiivista päättelyä eli päättelyä erityisestä yleiseen ohjelman tuottamiseen. Lähtökohdaksi otetaan joukko esimerkkejä, jotka kuvaavat ohjelman syötteitä ja niistä haluttuja tulosteita. Esimerkkien lisäksi usein annetaan myös taustatietoa, jota voidaan hyödyntää ratkaisun muodostamisessa. Esimerkkeinä annetaan myös yleensä negatiivisia esimerkkejä, jotka kuvaavat ohjelman ei-toivottua toimintaa. Esimerkkien avulla induktioalgoritmi tuottaa automaattisesti ohjelman. Esimerkit eivät koskaan voi olla täysin kattavia ja siksi lopputuloksena saatu ohjelmakaan ei välttämättä ole täydellinen. Ohjelmaa voidaan kuitenkin testata ja saatujen tulosten perusteella lisätä uusia esimerkkejä. Tämän jälkeen induktioalgoritmi ajetaan läpi uudelleen ja näin saadaan tarkennettu ohjelma, jota voidaan edelleen testata. Esimerkkejä ei siis edes oleteta täydellisiksi samoin kuin ei tuotettua ohjelmaakaan ennen kuin sen toiminta on käytännössä todettu virheettömäksi [Par97, BeG95].

4 Logiikkaohjelman synteesi

Lähtökohtana induktiivisessa logiikkaohjelmoinnissa on kaikkien mahdollisten logiikkaohjelmien joukko $\mathcal{P}$ sekä joukko esimerkkejä E. Esimerkit jakautuvat ohjelman haluttua toimintaa kuvaaviksi positiivisiksi esimerkeiksi E^+ ja ei-toivottua toimintaa kuvaaviksi negatiivisiksi esimerkeiksi E^-. Yleensä myös käytettävissä on taustatietoa B. Taustatieto ja esimerkit esitetään logiikkaohjelmana. Tuotettavalta ohjelmalta eli hypoteesilta H vaaditaan, että sen täytyy olla tarpeellinen, täydellinen ja ristiriidaton. Ohjelma on *tarpeellinen*, jos kaikki positiiviset esimerkit eivät loogisesti seuraa suoraan annetusta taustatiedosta, merkitään $B \nvdash E^+$. Ohjelma on *täydellinen*, jos se kattaa kaikki positiiviset esimerkit, eli $H \vdash e$, kaikille $e \in E^+$. Ohjelma on *ristiriidaton*, jos se ei kata yhtään negatiivista esimerkkiä, eli $H \nvdash e$, kaikille $e \in E^-$. Induktiivinen päättely ei ole täydellisen pitävää. Määritelmän täyttäviä ohjelmia saattaakin olla useita. Näistä induktiivinen logiikkaohjelmointijärjestelmä tuottaa lopputulokseksi yhden [BeG95].

Jos esimerkiksi tavoitteena on pikajärjestämisalgoritmin automaattinen synteesi, voidaan taustatietona antaa joukko mahdollisesti tarvittavia lauseita

$$B = \begin{cases} part(X, [], [], []). \\ part(X, [H|T], [H|S1], S2) \text{ :- } H < X, part(X, T, S1, S2). \\ part(X, [H|T], S1, [H|S2]) \text{ :- } H \geq X, part(X, T, S1, S2). \\ app([], L1, L2). \\ app([H|T], L, [H|R]) \text{ :- } app(T, L, R). \end{cases}$$

Positiivisina esimerkkeinä annetaan joukko järjestämisen alku- ja lopputilannetta kuvaavia esimerkkejä ja negatiivisina esimerkkeinä väärin menneitä järjestämisiä vastaavasti

$$E^+ = \begin{cases} qsort([], []). \\ qsort([0], [0]). \\ qsort([6, 1, 5], [1, 5, 6]). \\ \ldots \end{cases}$$

$$E^- = \begin{cases} qsort([],[]). \\ qsort([0],[]). \\ qsort([1,0],[0,1]). \\ qsort([7,2,9],[9,2,7]). \\ \ldots \end{cases}$$

Lopputulokseksi saadaan ohjelma

$$H = \begin{cases} qsort([],[]). & \\ qsort([H|T],S) \text{ :- } & part(H,T,L1,L2), \\ & qsort(L1,S1), \\ & qsort(L2,S2), \\ & app(S1,[H|S2],S). \end{cases}$$

Useat induktiiviset logiikkaohjelmointijärjestelmät pystyvät oppimaan pikajärjestämisen mainittua taustatietoa käyttäen vähintään kymmenestä esimerkistä [MuD94].

Induktiiviset logiikkajärjestelmät jaetaan karkeasti ottaen kahteen ryhmään, empiirisiin ja interaktiivisiin järjestelmiin. Interaktiiviset järjestelmät tarvitsevat ohjelman tuottamiseen oraakkelia, joka määrittelee välitulosten oikeellisuuden. Käytännössä oraakkelina toimii järjestelmän käyttäjä. Interaktiiviset järjestelmät myös yleensä muodostavat logiikkaohjelman alkaen esimerkeistä yleistämällä. Koska interaktiiviset järjestelmät voivat hyödyntää oraakkelia, voivat ne tehdä raskaampia oletuksia ja usein pystyvät tuottamaan tarkempia lopputuloksia vähemmällä tietomäärällä. Interaktiiviset järjestelmät yleensä tuottavat uusia predikaatteja ja pystyvät oppimaan useamman predikaatin määritelmän kerralla. Toisaalta niiden toiminta nojaa pitkälti käyttäjän aktiiviseen osallistumiseen ohjelman tuottamiseseen [LaD93].

Empiiriset järjestelmät vaativat suuremman joukon esimerkkejä tai taustatietoa ohjelman tuottamiseen kuin interaktiiviset järjestelmät. Myös jotkut empiiriset järjestelmät vaativat oraakkelin apua, mutta niissä käyttäjän osallistuminen voidaan rajoittaa aivan prosessin alkuvaiheeseen ja käyttäjän ei tarvitse tulkita kokonaisia sääntöjen paikkansapitävyyttä vaan pelkkien lisäesimerkkien, joita järjestelmä tuottaa tarpeen mukaan. Empiiriset järjestelmät pystyvät yleensä oppimaan vain yhden predikaatin määritelmän kerrallaan. Empiiristen järjestelmien etuna on mahdollista alkuvaihetta lukuun ottamatta automaattinen lopputuloksen indusointi. Tämä

on huomattava etu, sillä kymmeniä lauseita sisältävän ohjelman indusointi saattaa kestää pitkään [LaD93].

Empiirisissä järjestelmissä lähtökohta on interaktiivisiin järjestelmiin verrattuna käänteinen eli lauseiden muodostaminen aloitetaan tyhjästä lauseesta, johon yksi kerrallaan lisätään literaaleja [LaD93].

5 Induktiiviset logiikkaohjelmointikielet

Predikaattien määritelmien oppiminen voidaan ajatella etsimisongelmaksi. Kaikkien mahdollisten lauseiden joukosta yritetään löytää tarkoitukseen sopiva jonkin laatukriteerin mukaan. Lauseiden joukko on hyvin suuri, mikä tekee etsimisen työlääksi. Siksi yleensä induktiiviset logiikkaohjelmointijärjestelmät vaativat mahdollisten lauseiden joukon rajoittamista jollain tavalla. Rajoitusten esittäminen tapahtuu järjestelmistä riippuen eri tavoilla. Näitä eri menetelmiä kutsutaan yhteisesti nimellä *kielen esiviritys* (language bias) [LaD93, Tau94].

Kielen esiviritys on käytettävien esimerkkien ja taustatiedon jälkeen kolmas lopputuloksen määrittävä tekijä. Esimerkkien, taustatiedon ja kielen esivirityksen esittämiseen käytettävää kieltä voidaan kutsua *induktiiviseksi logiikkaohjelmointikieleksi.* Koska esimerkit ja taustatieto esitetään Prolog-kielellä, on induktiivinen logiikkaohjelmointikieli käytännössä kielen esivirityksen esittämiseen käytetty kieli. Induktiivinen logiikkaohjelmointijärjestelmä voidaan ajatella induktiivisen logiikkaohjelmointikielen kääntäjäksi, joka tuottaa Prolog-kielistä ohjelmakoodia lopputuloksenaan [BeG95].

Yksi tapa kielen esivirityksen esittämiseksi on *lausejoukot* (clause sets) [BeG95]. Lausejoukkojen esitystapa on lähellä Prologin kielioppia. Ainoa lisäys Prologiin on aaltosulut, joita käytetään esittämään vaihtoehtoisia osia. Aaltosuluilla erotelluista lauseista mikä tahansa tai kaikki voivat tulla mukaan lopulliseen ohjelmaan. Aaltosuluilla voidaan myös erotella vaihtoehtoisia literaaleja yksittäisen lauseen sisällä. Vaihtoehtoiset osat voivat myös sisältää muita vaihtoehtoisia osia. Esimerkiksi esiviritys

```
q(X) :- b(X).
{p(X) :- {q(X), a(X,Y)}.
q(X) :- c(X,Z), {q(Z)}}.
```

määrittelee mahdollisiksi ohjelmiksi muun muassa ohjelmat

```
q(X) :- b(X).
p(X) :- q(X), a(X,Y).
q(X) :- c(X,Z), q(Z).
```

ja

```
q(X) :- b(X).
p(X) :- a(X,Y).
q(X) :- c(X,Z).
```

Lausejoukkojen avulla määritelty kielen esiviritys rajaa hyvin voimakkaasti lopputulosta eli kyseessä on vahva esiviritys. Lausejoukkojen avulla on helppo kuvata jonkinlainen hyvä arvaus lopputuloksena haluttavasta ohjelmasta. Induktiivisen logiikkaohjelmointijärjestelmän tehtävänä on yrittää tuottaa esimerkkien mukaan oikein toimiva ohjelma arvauksen pohjalta. Näin ohjelmoija voi jättää vaikeiden algoritmien yksityiskohtaisen toteutuksen induktiivisen logiikkaohjelmointijärjestelmän huoleksi [BeG95].

6 Yhteenveto

Induktiivinen logiikkaohjelmointi on koneoppimisen menetelmä logiikaohjelmien automaattiseen tuottamiseen ohjelman haluttua toimintaa kuvaavien esimerkkien pohjalta. Tuotettavat ohjelmat voivat olla tarkoitukseltaan jotain ilmiötä kuvaavia malleja tai puhtaasti ajettaviksi tarkoitettuja ohjelmia. Jälkimmäisessä tapauksessa induktiivinen logiikkaohjelmointi voidaan nähdä ohjelmistotuotannon apuvälineenä, jonka avulla voidaan hankalien algoritmien toteuttamista automatisoida.

Lähteet

BeG95 Bergadano, F. ja Gunetti, D., *Inductive Logic Programming: From Machine Learning to Software Engineering*. The MIT Press, 1995.

ClM95 Clocksin, W. F. ja Mellish, C. S., *Programming in Prolog (2nd ed.)*. Springer-Verlag New York, Inc., 1984.

LaD93 Lavrac, N. ja Dzeroski, S., *Inductive Logic Programming: Techniques and Applications*. Routledge, 1993.

MuD94 Muggleton, S. ja De Raedt, L., Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20, sivut 629–679.

Par97 Partridge, D., The case for inductive programming. *IEEE Computer*, 30,1(1997), sivut 36–41.

StS86 Sterling, L. ja Shapiro, E. *The art of Prolog: advanced programming techniques*, sivut 2–18. MIT Press, 1986.

Tau94 Tausend, B., Biases and their effects in inductive logic programming. *Proceedings of the 7th European Conference on Machine Learning*, Bergadano, F. ja De Raedt, L., toimittajat, osa 784. Springer-Verlag, 1994, sivut 431–434, URL `citeseer.ist.psu.edu/tausend94biases.html`.