Logiikkaohjelmointi

Petri Kärki

Helsinki 2. huhtikuuta 2004

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Ohjelmointikielten periaatteet

Sisältö

1 Johdanto

Imperatiivisessa ohjelmoinnissa ohjelmoija kertoo algoritmin jolla ongelma ratkaistaan. Logiikkaohjelmoinnissa ohjelmoija kuvaa ongelman ja logiikkaohjelmointikieli luo algoritmin joka ratkaisee ongelman. Tästä johtuen logiikkaohjelmoinnissa ohjelman kehitys koostuu pääasiassa ongelman kuvaamiseen tarvittavan tietomäärän kasaamsisesta. Logiikkaohjelmoinnista on hyötyä koska logiikkaohjelmointikielillä tehdyt ohjelmat ovat helpompia laajentaa jos sovellusalueesta on saatu uutta tietoa. Lisäksi logiikkaohjelmointikielissä voidaan rakaisuun johtanut päättelyketju tallettaa myöhempää tutkimista varten ja näin on ohjelmoijalla mahdollisuus nähdä miten ohjelma on ratkaissut ongelman [GeGi85]. Tästä johtuen logiikkaohjelmointia kutsutaan myös deklaratiivseksi ohjelmoinniksi. Logiikkaohjelmoitikieliä käytetään enimmäkseen formaalissa spesifikaatiossa sekä väittämien todistuksissa.

2 Logiikkaohjelmoinnin periaatteet

Logiikkaohjelmoinnissa ohjelmoija kuvaa ongelma-alueen joukkona aksioomia (faktoja sekä sääntöjä). Ohjelman suoritus muodostuu siitä että ohjelmoija esittää ongelmaalueeseen liittyvän kyselyn (goal) johon logiikkaohjelmointikielen implementaatio etsii aksioomien pohjalta vastauksen. Melkein kaikissa kielissä aksioomat on kirjoitettu standardi muodossa. Tätä muotoa kutsutaan Horn klausuuliksi.

$$H \leftarrow B_1, B_2, ..., B_n$$

Horn klausuuli koostuu yhden termin H sisältävästä päästä (head) sekä termit B_i sisältävästä rungosta (body). Informaalisesti klausuuli kertoo että termi H toteutuu mikäli kaikki termit B_i toteutuvat. Jos klausuulissa on ainoastaan vasen puoli eli $n = 0$ on kyseessä fakta. Termit koostuvat vakioista sekä predikaateista. Vakion arvo on joko tosi tai epätosi ja muutumaton. Predikaatit kuvaavat muuttujan arvon tai ongelmajoukon alkion todeksi tai epätodeksi [Scot00].

Suurin osa logiikkaohjelmointikielistä perustuu ensimmäisen kertaluvun logiikkaan eli predikaattilogiikkaan. Matematiikassa yleensä ja niin myös predikaattilogiikassa on monta tapaa sanoa sama asia. Lauseiden monimuotoisuus haittaa automaattista päättelyä. Tämän takia jos halutaan suoritttaa automaattista laskentaa predikaattilogiikan kaavoilla täytyy ne ensin muuttaa edellä esitettyyn klausuuli muotoon. Muutoksen aikana eliminoidaan lausekkeesta eksistentiaali ja universaali kvanttorit. Kaikkia predikaattilogiikan kaavoja ei voida kuitenkaan muuttaa klausuuli muotoon. Tästä johtuen logiikkaohjelmointikielet ovat ilmaisuvoimaltaan heikompia kuin predikaattilogiikka.

Logiikkaohjelmoinnissa laskenta kuvataan loogisena päättelyketjuna. Logiikkaohjelmointikielen toteutus yhdistää lauseita tunnetusta aksiooma joukosta uusiksi lauseiksi resoluutioperiaatteen avulla. Esimerkiksi jos meillä on klausuuli K_1 jonka

vasemmalla puolella on termi T ja toinen klausuuli K_2 missä termi T on oikealla puolella voidaan termi T korvata klausuulissa K_2 klausuulin K_1 rungolla.

Logiikkaohjelmointikieli voi vastata kyselyyn kahdella tavalla. Voidaan käyttää eteenpäin ketjutusta (forward chaining) jossa jo olemassa olevista klausuuleista yritetään johtaa kysely. Toinen vaihtoehto on käyttää taaksepäin ketjutusta (backward chaining). Taaksepäin ketjutuksessa lähdetään kyselystä ja yritetään hajottaa se tietokannassa sijaitseviksi klausuuleiksi. Yleensä taaksepäin ketjutus löytää ratkaisun kyselyyn nopeammin, mutta jos sääntöjen lukumäärä on suuri ja faktoja on vähän voi eteenpäin ketjutus olla tehokkaampaa.

Automatisoitu päättely on epädeterministinen prosessi. Tästä johtuen voidaan joutua käymään läpi monta eri päättelypolkua ennen haluttu ratkaisu saavutetaan. Joskus on haluttavaa eliminoida osia etsintäavaruudesta pois ja tutkia vain jäljelle jäävät polut. Eri logiikkaohjelmointikielet antavat eri tapoja ohjelmoijalle suorittaa tällaista hallintaa. Jossain kielissä kuten Prologissa ohjelmoija voi implisiittisesti vaikuttaa etsinnän etenemiseen klausuulien ja termien järjestyksellä sekä eksplisiittisesti käyttämällä syntaktisia ominaisuuksia kuten ! operaatiota. Jotkin kielet kuten MRS tarjoavat laskennan ohjaukseen yleisen metatason ohjauskielen [GeGi85].

3 Eräitä logiikkaohjelmointikieliä

Tässä kappaleessa tarkastellaan kahta logiikkaohjelmointikieltä hieman tarkemmin. Kumpikaan tarkasteltavista kielistä ei ole puhdas logiikkaohjelmointikieli vaan ne sisältävät piirteitä imperatiivisista ohjelmointikielistä.

3.1 Prolog

Prolog on yleisin logiikkaohjelmointikieli. Prolog nimi tulee sanoista programming in logic. Prolog kielen kehitti AI tutkija Alan Colmerauer 1970 luvun alussa. Nykyään Prologista on olemassa monia erilaisia murteita. Prolog kielen tietokanta koostuu klausuuleista [MacL83]. Prologissa tietokannan läpikäynti toteutetaan siinä järjestyksessä missä klausuulit on ohjelmaan kirjotettu. Klausuulien kirjoitusjärjestyksellä on vaikutus ohjelman suoritusaikaan. Jossain tilanteissa klausuulien tietynlainen järjestys voi aiheuttaa sen että ohjelman suoritus ei pääty koskaan.

Prologissa termit koostuvat vakioista muuttujista sekä rakenteista. Rakenteet koostuvat vakiosta jota kusutaan funktoriksi sekä parametrilistasta. Prologissa ei ole taulukoita vaan niiden tilalla käytetään listoja. Ainoastaan listan ensimmäiseen alkioon päästään käsiksi. Kuten jo aikaisemmin mainittiin Prolog käyttää ! eli leikkaus operaatiota etsintäavaruuden rajaamiseen. Leikkaus operaation avulla voidaan Prologissa toteuttaa $if...then...else$ rakenne. Leikkaus operaatiota käytetään Prologissa myös silmukkarakenteen toteuttamiseen. Prologissa aritmeettiset operaatiot suorite-

taan Prologin sisäänrakennetun funktorin avulla koska normaalit aritmeettiset operaatiot toiminta poikkeaa totutusta.

Prolog vastaa kyselyyn käyttäen taaksepäin ketjutusta (backward chaining). Eli toisinsanoen prolog etsii ne klausuulit jotka tuottavat halutun kyselyn. Kyselyyn liittyvä etsintä voidaan kuvata puurakenteena.Prolog tulkki käy läpi tätä puuta syvyyssuunnassa vasemmalta oikealle. Saadessaan kyselyn Prolog tulkki aloittaa tietokannan läpikäynnin etsimällä sellaista klausuulia K jonka vasen puoli voidaan samastaa kyselyn kanssa. Samastus tarkoittaa sitä että kahden klausuulin A ja B muuttujille annetaan arvot siten että $A = B$. Tämän jälkeen se käsittelee vasemmalta oikealle klausuulin K rungon jokaisen termin rekursiivisesti omana kyselynään. Näitä kyselyitä kutsutaan alikyselyiksi (subgoals). Jos joku näistä kyselyistä epäonnistuu tulkki lopettaa klausuulin käsittelyn ja etsii toisen klausuulin jonka vasen puoli voidaan samastaa kyselyn kanssa ja yrittää käsitellä tämän kuten edellä. Tätä prosessia jossa palataan edeltäneeseen kyselyyn kutsutaan perääntymiseksi (backtracking)[Scot00].

Seuraavana oleva Prolog kielinen ohjelma laskee kahden luvun suurimman yhteisen tekijän.

```
gcd(A,B,GCD) :- A = B, GCD = A.
gcd(A,B,GCD) :- A < B, NB is B - A, gcd(A,NB,GCD).
gcd(A,B,GCD) :- A > B, NA is A - B, gcd(NA,B,GCD).
```

Ohjelma joka laskee suurimman yhteisen tekijän on aika lyhyt ja selkeä, jos sitä vertaa imperatiivisellä kielelä kuten C:llä tehtyyn ratkaisuun.

3.2 Gödel

Gödel on deklaratiivinen yleiskäyttöinen logiikkaohjelmointikielten perheeseen kuuluva kieli. Gödel on nimetty Kurt Gödelin tunnetun matemaatikon mukaan.Gödel kielen ovat kehittäneet Bristolin Yliopistossa Antony Bowers, John Lloyd sekä Pat Hill. Kielen kehittäjien tarkoituksena oli tehdä Prologia parempi logiikkaohjelmointikieli. Gödelin suunnittelun tavoitteena oli että kielellä olisi sama toiminnallisuus ja ilmaisuvoima, mutta paljon kehittyneempi deklaratiivinen semantiikka kuin Prologilla. Näin oli tarkoitus pienentää kuilua käytännön logiikkaohjelmointikielten ja logiikka ohjelmoinnin teorian välillä. Gödel tukee äärettömän tarkkuuden kokonaislukuja, äärettömän tarkkuuden liukulukuja sekä äärettömän tarkkuuden murtolukuja. Gödel kieli tukee myös meta ohjelmointia. Meta ohjelma on ohjelma joka muuttaa tai luo muita ohjelmia. Gödel on vahvasti tyypitetty kieli, jonka tyyppijärjestmä perustuu monilajittelu logiikkaan parametrisella polyymorphismilla. Vahvan tyypityksen ansiosta kääntäjä voi optimoida koodia ja syntaksivirheet löytyvät jo käänösaikana. Parametrisella polymorphismilla tarkoitetaan että parametroitavalle tyypille voidaan kielessä valita parametrin tyyppi kun parametroitavasta tyypistä

luodaan ilmentymä. Esimerkiksi taulukosta voidaan luoda ilmentymänä kokonaisluku taulukko tai kirjain taulukko. Gödelissä on moduuli järjestelmä joka perustuu Modula-2 esiintyvään moduuli järjestelmään. Gödel sisältää myös joustavan laskenta säännön. Siinä termit joista kysely koostuu voidaan suorittaa missä järjestyksessä tahansa eikä vain vasemmalta oikealle kuten Prologissa. Gödelissä on etsintäavaruuden kontrollointiin prologia tehokkaampia mekanismeja kuten esimerkiksi Commit joka vastaa Prologin ! operaatiota [HiLl00].

Edellisessä luvussa oli esitetty Prolog kielinen ohjelman joka laski kahden luvun suurimman tekijän. Tässä sama kahden luvun suurimman yhteisen tekijän laskeva moduuli Gödel kielellä toteutettuna.

```
MODULE      GCD.

IMPORT      Integers.

PREDICATE   Gcd : Integer * Integer * Integer.

Gcd(i,j,d) <-
            CommonDivisor(i,j,d) &
            ~ SOME [e] (CommonDivisor(i,j,e) & e > d).

PREDICATE   CommonDivisor : Integer * Integer * Integer.

CommonDivisor(i,j,d) <-
            IF (i = 0 \/ j = 0)
            THEN
              d = Max(Abs(i),Abs(j))
            ELSE
              1 =< d =< Min(Abs(i),Abs(j)) &
              i Mod d = 0 &
              j Mod d = 0.
```

Gödelin suurimmanyhteisen tekijän laskeva ohjelman koodi on paljon isompi johtuen osaksi siitä että Gödeliin on otettu enemmän käskyjä kuten if...then...else rakenne. Kielen rakenteiden lisääminen näkyy siinä että Gödelin ratkaisu alkaa näyttää jo yhtä selkeältä kuin imperatiivisella kielellä tehty.

4 Yhteenveto

Logiikkaohjelmointiin ja logiikkaohjelmointikieliin liitty vielä tälläkin hetkellä monia heikkouksia. Nykyaikaiset logiikkaohjelmointikielet ovat rajoittuneita. Horn klausuuleilla ei voi ilmaista kaikki predikaattilogiikan ongelmia. Joukko Horn klausuuleita kertoo vain mitkä asiat ovat totta. Jos joukko ei sisällä kaikkia asioita mitkä oletetaan tosiksi voi olla, että kysely T epäonnistuu vain koska käytössäoleva Horn klausuulien joukko ei sisällä tarpeeksi informaatiota T:n todistamiseen. Logiikkaohjelmointi kielten toteutukset ovat hitaita ja ne eivät sovellu kaikkiin tehtäviin kuten esimerkiksi graafisen I/O:n ohjelmointiin.

Viitteet

GeGi85 Michael Genesereth, Matthew Ginsberg, Logic Programming *Communications of the ACM* 28, 9 (1985), 933-941.

HiLl00 Patricia Hill, John Lloyd, *The Gödel Programming Language*, MIT Press, 1994.

MacL83 Bruce MacLennan, *Principles of Programming languages*, Holt-Saunders International, 1983.

Scot00 Michael Scott, *Programming Language Pragmatics*, Morgan Kaufmann Publishers, 2000.