Matematički fakultet

Projekat iz predmeta Automatsko rezonovanje

Školska 2020/2021

Nelson-Oppen algoritam za kongruentno zatvorenje

Autori:
Nikola Stojević 1111/2018

Profesor:
Dr. Milan Banković
Asistent:
Ognjen Kocić

Sadržaj

1 Uvod

U okviru kursa Automatsko rezonovanje bavili smo se prvo problemom ispitivanja zadovoljivosti u iskaznoj logici, a zatim problemom ispitivanja zadovoljivosti u logici prvog reda. Simbol '=' se može u logici prvog reda interpretirati na razne načine, ali nama najvažnije interpretacije jesu one u kojima se ovaj simbol posmatra baš kao relacija jednakosti. Za sve modele koji zadovoljavaju prethodno pomenutu interpretaciju simbola '=' kažemo da su normalni modeli. Oni zadovoljavaju aksiome jednakosti: refleksivnost, simetričnost, tranzitivnost, kao i kongruentnost. Sintaksno-deduktivni sistem za jednakost opisuju Birkohofova pravila. Neka jednakost važi samo ukoliko ona može da se izvede iz početnih pretpostavki primenom Birkohofovih pravila. Ukoliko su sve jednakosti bazne, odnosno nemaju slobodnih promenljivih, onda nije potrebno koristiti pravilo instancijacije iz Birkohofovih pravila, tako da imamo fragment logike prvog reda koji je odlučiv.

Za ovakav pristup mogu se izgraditi procedure specijalizovane za rezonovanje u prisustvu jednakosti. Procedure odlučivanje za bazne jednakosti se obično zasnivaju na kongruentnom zatvorenju. Jedna od takvih jeste procedura odlučivanja Nelson-Oppen koja će biti detaljnije opisana u nastavku.

2 Nelson-Oppen procedura

Relacija ˜je *kongruencija* na skupu D u odnosu na jezik L ako je relacija ekvivalencije na domenu D saglasna (kongruentna) sa svim funkcijskim simbolima jezika L, tj. za svako f važi da ako s1 ˜t1, ..., sn ˜tn, tada f(s1,..., sn) ˜f(t1,..., tn). *Kongruentno zatvorenje* relacije je najmanja kongruencija koja sadrži polaznu relaciju.

Nelson-Oppen algoritam radi na tome da dobijemo kongruentno zatvorenje. Počinjemo od prazne relacije i dodajemo jednu po jednu jednakost proširujući time relaciju kongruencije. Svaka od kongruencija se predstavlja klasama ekvivalencije.

Za predstavljanje klasa ekvivalencija u algoritmu se koristi struktura podataka *union-find*, koristimo obrnuto drvo (pokazivači idu od listova ka korenu). Da bi postupak bio što efikasniji, potrebno je da drvo bude što pliće. Ukoliko imamo dva terma za koja važi da imaju isti koren, onda to znači da ta dva terma jesu u istoj klasi ekvivalencije. Postupak objedinjavanja dve klase ekvivalencije je efikasan, samo treba preusmeriti pokazivač jednog korena na drugi, pri čemu treba voditi računa da stablo ostane što pliće.

Predstavimo sa E = {s1 = t1, ..., sn = tn} skup svih baznih jednakosti za koje pravimo relaciju kongruencije. Neka je T skup svih baznih termova iz E, njihovih podtermova i podtermova od podtermova, itd. (što znači da je skup zatvoren za podtermove), a skup T može da sadrži i još neke termove (termove za koje ćemo da ispitujemo da li se nalaze u istoj klasi ekvivalencije). Kada smo pokupili sve termove, za ovaj algoritam je potrebno je definisati sledeće funkcije i skupove:

- **USE(t)** označava skup svih termove skupa T čiji je direktan podterm term t.
- **FIND(t)** vraća kanonskog predstavnika klase ekvivalencije kojoj pripada term t, tj. vraća koren drveta u kome se nalazi term t.
- **UNION(s,t)** spaja klase ekvivalencija terma s i terma t. Pri spajanju voditi računa o dubini drveta, odnosno, preusmeravati pokazivač od plićeg ka dubljem drvetu.
- **CONG(s,t)** označava da s i t imaju iste funkcijske simbole i njihovi podtermovi pripadaju istoj klasi ekvivalencije. Tačnije neka je s oblika f(s1, ..., sn) i t oblika f(t1, ..., tn), pri čemu je find(si) = find(ti).

```
function computeClosure(E,T)
begin
    // na pocetku svaki term je predstavnik svoje klase i jedini term u
        klasi
    foreach t from T
        find(t):=t
    //spajamo termove koji pripadaju baznim jednakostima u istu klasu
    foreach s = t from E
        if(find(s) != find(t))
            merge(s, t)
end
```

```
function merge(s,t)
begin
    //pronadji termove u kojima se koriste termovi s i k i svi termovi
    //koji se nalaze u njihovim klasama ekv.
    Ts = U{use(u)| find(u)=find(s)}
    Tt = U{use(u)| find(u)=find(t)}
    union(s,k)
    foreach s' from Ts and t' from Tt
        if(find(s')!=find(t') and cong(s',t'))
            merge(s',t')
end
```

3 Implementacija

Implementacija algoritma obuhvata šest fajlova: lexer.lpp, parser.ypp, unionFind.hpp, unionFind.cpp, fol.hpp, main.cpp. Proći ćemo kroz svaki od njih i objasniti najvažnije funkcionalnosti koje oni pružaju.

3.1 lexer

Lekser potreban za čitanje sa ulaza. Funkcijski simboli jesu stringovi koji počinju malim slovom. Jedini predikatski simbol koji može da se pročita jeste '=' koje nam je potrebno za čitanje skupa jednakosti koji predstavljaju ulaz u algoritam. Ostali dozvoljeni simboli jesu: ( ) { } ; , i razmaci.

3.2 parser

U okviru ovog fajla nalazi se parser potreban za čitanje sa ulaza. Ulaz je formata koji izgleda ovako '{' setFormulas '}' '{' formula '}' ';' , odnosno {jednakost 1 , jednakost 2 , ...jednakost n }{jednakost n+1 };. Prva vitičasta zagrada jeste skup E opisan u odeljku 2. Druga vitičasta zagrada jeste jedna- kost za koju želimo da dokažemo da se može izvesti iz skupa E. Termovi iz ove vitičaste zagrade takode ulaze u skup T .

```
//niz jednakosti potrebnih za algoritam
vector<Formula>* parsed_set_of_formulas;
//jednakost koja zeli da se dokaze da vazi
Formula parsed_formula;
```

3.3 fol

Ovo je zaglavlje potrebno za implementaciju logike prvog reda.

BaseTerm je apstraktna klasa koja se koristi za predstavljanje termova. Jedini termovi koji su nam potrebni u okviru naše implementacije jesu funkcijski termovi predstavljeni klasom FunctionTerm. Svaki funkcijski term se sastoji iz funkcijskom simbola i iz niza njegovih podtermova. Funkcija equalTo je funkcija koja proverava da li su dva terma jednaka. Dva terma su jednaka ukoliko su im isti funkcijski simboli i ukoliko su im isti odgovarajući podtermovi.

BaseFormula je apstraktna klasa koja se koristi za predstavljanje formula. Jedine formule potrebne za našu implementaciju jesu atomi, i to jednakosti predstavljena klasom *Equality*. Svaka jednakost se sastoji iz predikastskog simbola '=' i iz dva podterma.

Kako su nam u daljoj implementaciji potrebni skupovi termova i mapa čiji su ključevi termovi, bilo je potrebno redefinisati operator $<$. Za dva terma s i t kažemo da je s manji od t ukoliko je funkcijski simbol leksikografski manji od funkcijskog simbola terma t. Ukoliko su funkcijski simboli isti, prvi term je manji od drugog ako ima manji broj podtermova. Ako je broj podtermova jednak, onda uporedujemo odgovarajuće podtermove termova s i t. Prvi term je

manji ako naidjemo da mu je prvi podterm manji od od odgovarajućeg podterma drugog terma. Ovde su definisani i skup termova *TermSet* i mapa za predstavljanje use skupova *UseMap*.

```
//C je komparator koji koristi operator <
typedef set<Term, C> TermSet;
typedef map<Term, TermSet, C> UseMap;
```

3.4 unionFind

Funkcija *findPosition(Term s)* pronalazi poziciju u vektoru datog terma s. Ova funkcija će vratiti -1 ukoliko ne pronadje dati term u okviru vektora.

Funkcija *findRootOfTerm(Term t)* pronalazi poziciju u vektoru od ka- nonskog predstavnika klase ekvivalencije u okviru koje se nalazi term t. Krećemo se pokazivačima na roditelje sve dok ne naidemo da neki N ode pokazuje na samog sebe, što označa da je on koren stabla. Vraća -1 ako term t ne postoji u vektoru.

Funkcija *findAllRoots()* pronalazi sve kanonske predstavnike klasa ekvivalencije i vraća skup termova koji im odgovaraju.

Funkcija *findAllRootNodes()* je ista kao prethodna samo što vraća skup Node-ova koji im odgovaraju.

Funkcija *unionOfSets(Term firstTerm, Term secondTerm)* spaja dve klase ekvivalencija. Funkcija zapravo pronalazi kanonske predstavnike odgovarajućih klasa ekvivalencije. Funkcija će preusmeriti pokazivač na roditelja korena plićeg stabla na pokazivač korena dubljeg stabla, a ukoliko su dubine jednake, onda će preusmeriti pokazivač roditelja od drugog na prvog. U tom slučaju je potrebno povećati dubinu stabla za jedan.

Funkcija *findTermsFromTheSameSet(Term t)* pronalazi sve termove koji se nalaze u okviru iste klase ekvivalencije kao term t i vraća njihov skup. Funkcija radi tako što prolazi kroz ceo vektor i proverava za svaki element da li se kanonski predstavnik njegove klase ekvivalencije poklapa sa kanonskim predstavnikom klase ekvivalencije u okviru koje se nalazi term t. Funkcija ne ubacuje u skup i sam term t.

3.5 main

U ovom fajlu se nalaze potrebne funkcije za implementaciju algoritma Nelson-Oppen.

Funkcija *getTerms(vector <Formula >* vf, Formula f, TermSet & allTerms)* kupi sve termove iz vf , a zatim iz f i ubacuje ih u skup termova allTerms koji je zatvoren za podtermove. To radi pomoću funkcije koja kupi termove iz jedne formule tako što prvo pokupi termove iz terma sa leve strane jednakosti, a zatim iz terma sa desne strane jednakosti. Njoj pomaže funkcija koja kupi termove iz terma. Ta funkcija u skup allTerms prvo ubaci term za koji je pozvana, a zatim

poziva samu sebe za svaki svoj podterm. Izlaz iz rekurzije je kad naidemo na simbol konstante, odnosno, kada naidemo na term koji nema podtermove.

Funkcija *getUseMap(UseMap & um, TermSet & t)* je funkcija koja za svaki term iz t proverava u okviru kojih termova se on koristi tj. gde se nalazi kao direktan podterm i ubacuje u mapu um par: term i skup termova iz t u okviru kojih se ključ koristi.

Funkcija *cong(Term firstTerm, Term secondTerm, UnionFind & u)* odgovara opisanoj funkciji iz odeljka 2. Vraća true ako su funkcijski simboli termova firstTerm i secondTerm isti i ukoliko se kanonski predstavnici odgovarajućih podtermova poklapaju.

Funkcija *merge(Term s, Term t, UnionFind & u)* odgovara opisanoj funkciji iz odeljka 2. Pomoću funkcije *getSetsForMerge(Term s, UnionFind & u)* pronalaze se skupovi Ts = U { use(s') | find(s') == find(s) } i Tt = U { use(t') | find(t') == find(t) }. Ova pomoćna funkcija uzima iz use mape odgovarajuće union find strukture u skup termova u okviru kojih se koristi term s, a zatim za sve termove koji se nalaze u istoj klasi ekvivalencije kao i term s pronalazi skupove termove iz use mape u okviru kojih se oni koriste. Vraća odgovarajuću uniju termova. Vratimo se na glavnu funkciju merge. Kada je pronašla skupove, ova funkcija spaja klase ekvivalencije za termove s i t, a zatim za svaki par termova iz pronadjenih skupova proverava da li je potrebno spojiti njihove klase ekvivalencija. Spaja ih ako su im kanonski predstavnici klasa ekvivalencija u kojima se nalaze različiti i ako funkcija cong pozvana za ta dva terma vraća true.

Funkcija *cc(vector <Formula >* E, TermSet T, UnionFind & u)* odgovara opisanoj funkciji iz odeljka 2. Prolazi kroz sve jednakosti iz vektora E i poziva merge za njima odgovarajuće termove samo ukoliko važi da se nalaze u različitim klasama ekvivalencije.

Funkcija *checkEquality(vector <Formula >* E, Formula f)* koja proverava da li se termovi iz jednakosti f nalaze u okviru iste klase ekvivalencije i vraća true ako to i važi. Funkcija uzima termove iz E i f , kreira njima odgovarajuću UseMap, stvara UnionFind strukturu pomoću koje vodimo računa o klasama ekvivalencije, a zatim poziva funkciju cc koja će napraviti klase ekvivalencija. Kada funkcija cc završi svoj posao, glavna funkcija uzima levi i desni term iz jednakosti f , prolazi kroz sve klase ekvivalencije sve dok ne naide da su termovi iz jednakosti f u istim klasama ekvivalencije. Ako ne pronade, vraća false, što znači da se iz datog skupa jednakosti ne može zaključiti jednakost f .

4 Zaključak

Dalje primena je vezana za problem odlučivanja univerzalnog fragmenta EUF (Equality with Uninterpreted Functions).

1. Razmatra se validnost formula oblika ![x1...xn] P(x1,..., xn), gde P ne sadrži druge predikatske simbole osim = (može sadržati proizvoljne funkcijske simbole).

2. Negacijom i skolemizacijom se dobija bazna formula P'(c1,..., cn) (pokazujemo da je negacija nezadovoljiva iz čega zaključujemo da je polazna formula valjana).

3. Prevodjenjem u DNF, ispitivanje zadovoljivosti konjukcija oblika

   s1 = t1 & ... & sn =tn & s'1 != t'1 & ... & s'n' != t'n'

4. Za svaku konjukciju:

   - odredjuje se kongruentno zatvorenje za skup jednakosti E = {s1 = t1,..., sn = tn} i skup termova T={s1, t1,..., sn, tn, s'1, t'1,..., s'n', t'n'} i njihovih podtermova.
   - konjukcija je zadovoljiva ako za svaki par termova koji formiraju nejednakost važi da se ne nalaze u istoj klasi ekvivalencije

5. Ako nijedna konjukcija nije zadovoljiva početna formula je valjana.