Apstraktna interpretacija

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Ozren Demonja, Stefan Maksimović, Marko Crnobrnja
mi12319@alas.matf.bg.ac.rs, mi12078@alas.matf.bg.ac.rs, mi12024@alas.matf.bg.ac.rs

1. april 2017.

Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da ispoštujete!) kao i par tehničkih pomoćnih uputstava. Molim Vas da kada budete predavali seminarski rad, imenujete datoteke tako da sadrže temu seminarskog rada, kao i imena i prezimena članova grupe (ili samo temu i prezimena, ukoliko je sa imenima predugačko). Predaja seminarskih radova biće isključivo preko web forme, a NE slanjem mejla.

Sadržaj

1 Uvod

U protekle dve decenije se dosta toga promenilo u pogledu preformansi računara. Današnji kućni računari su jači nego najmoćniji superračunari iz 70-ih. U međuvremenu, kroz paralelizovanje i inovacije u hijerarhiji memorije superračunari sada postižu 10 do 100 teraflopa(eng. floating point operations per second). [2]

Glavni krivci za ovakvo poboljšanje u brzini računara su dva aspekta. Prvi, osnovna tehnologija prema kojoj se računari konstruišu je doživela izuzetan napredak koji počiva na predviđanjima Murovog zakona (eng. Moore's law) [6]. Drugi aspekt je paralelizam u nekoj svojoj formi [2].

Ova poboljšanja u snazi nisu došla bez problema. Kako je arhitektura postajala sve više i više kompleksna da bi mogla pratiti eksponencijalnu brzinu Murovog zakona, postajalo je sve teže i teže programirati. Većina vrhunskih programera je postala svesna potrebe da eksplicitno upravlja memorijom. U naporu da se poboljšaju preformanse pojedinačnih procesa, programeri su učili kako da ručno transformišu njihov kod tako da se efikasnije izvrši planiranje instrukcija na višeprocesorskom sistemu. [2]

U današnje vreme značajni deo koda u većini modernih kompajlera je posvećen optimizaciji generisanog koda. Često se dešava da ponašanje pri izvršavanju optimizovanog koda nesaglasno sa pre-optimizovanim ponašanjem koda, drugim rečima optimizacija je uticala kako na semantiku programa tako i na pragmatiku. Ovaj problem se često dešava zbog nedovoljne strogosti koja je bila primenjena na ispravnost dokaza optimitacije. Za programske jezike sa definisanom matematičkom semantikom postoji rastući skup alata koji obezbeđuju osnovnu za semantički korektnu transformaciju, jedan od tih alata je i apstraktna interpretacija. [1]

2 Apstraktna interpretacija

Kao što se vidi iz prethodnog poglavlja apstraktna interpretacija je tehnika za automatsku statičku analizu. Sastoji se od zamene preciznih elemenata programa sa manje detaljnim apstrakcijama. Apstrakcija dovodi do gubitka sigurnih informacija, što dovodi do nemogućnosti dovođenja zaključaka za sve programe. Apstraktna interpretacija omogućava da otkrijemo runtime greške, kao što su delenje sa 0, prekoračenje, itd, a takođe otkriva korišćenje zajedničkih promenjljivih i mrtvi petlji. [1]// Glavna prednost alata koji koriste apstraktnu interpretaciju je da se test obavlja bez iakve pripreme, baziran na kodu projekta. Ako se uporedi sa troškovima jediničnog testiranja, to predstavlja značajan argument. [1] //

2.1 Problem koji se rešava

Da bi se lakše shvatio problem prvo će mo pokazati dva ne programerska primera apstraktne interpretacije koja će služiti za uspostavljanje principa pristupa. //

Pretpostavimo da želimo da putujemo negde. Jedna od odluka koju moramo napraviti je da li želimo da hodamo, vozimo se ili letimo. Umesto da ovu odluku sprovodimo metodom pokušaja i greške, mi će mo

koristiti osobinu putovanja, udaljenost (koju možemo izmeriti na mapi) da odlučimo koji je najbolji način transporta. Mapa je apstraktna reprezentacija putovanja i merenjem rastojanja mi apstrahujemo sam proces putovanja. //

Drugi primer, malo više formalan, se gradi krisćenjem pravila zanak. Određujemo znak rezultata množenja. Ako se pitamo koji je znak

336 * (-398)

mi odmah znamo da je rezultat negativan. Bez da izvodimo množenje pa određujemo znak mi na osnovu pravila znaka znamo da će množenje pozitivnog i negativnog broja uvek proizvesti za rezultat negativan broj. Ovaj drugi primer je malo bliži apstraktnoj interpretaciji kod programiranja tako da će mo malo dublje zaći u njega. //

Da bi smo razumeli apstraktnu interpolaciju moramo da prebacimo zadatak u sledeću formu:

$$+ \times -\tag{1}$$

gde $\times$ predstavlja pravilo znaka pri mnozenju

$$\begin{gathered} 0 \times + = 0 \times - = + \times 0 = - \times 0 = 0 \\ + \times + = - \times - = + \\ + \times - = - \times + = + \end{gathered}\tag{2}$$

i onda izvodimo ove jednostavije izraze. Do sada nismo razmatrali korektnost interpretacije ali treba da bude jasno da mozemo dobiti potpuno tacne odgovor u oba primera. Ova situacija postaje mnogo nejasnija ako umesto mnozenja stavimo sabiranje. Prvih nekoliko redova ne predstavljaju neki problem //

$$\begin{gathered} 0 \pm + = + \pm 0 = +0 \\ 0 \pm - = - \pm 0 = -0 \\ + \pm + = + \\ - \pm - = - \end{gathered}\tag{3}$$

Ali ostatak je problematican:

$$\begin{gathered} + \pm - = ?? \\ - \pm + = ?? \end{gathered}\tag{4}$$

Ako bi stavili znak (0, +, -) a da ne znamo vrednosti u nekim slucajevima bi pogresili jer odgovor zavisi od vrednosti na koje se primenjuje. Kako mozemo da okarakterisemo pravi izbor za ??. Da bi mogli to da uradimo moramo da znamo koji znak u abstraktnom izracunavanju predstavlja:

$$0 = 0$$

$$\begin{aligned} + &= n | n > 0 \\ - &= n | n < 0 \end{aligned} \tag{5}$$

Onda je apstraktna kalkulacija tacna ako je pravi odgovor clan seta koji apstraktni odgovor predstavlja. Ako je ovo slucaj mi kazemo da je apstraktna interpretacija sigurna. Ako koristimo ?? da predstavimo cele brojeve, dobijamo sigurnu verziju sabiranja dodavanjem pravila:

$$s \pm ?? \quad = ?? \quad \pm \quad s \quad = ?? \pm ?? \quad = ?? gdes \quad \in \quad [0, -, +] \tag{6}$$

2.2 Koriscenje u racunarstvu

Kako je apstraktna interpretacija korisna u računarstvu? Mnogi tradicionalni optimizatori koji su bazirani na upravljanju tokom (eng. control flow) i na analizi toka podatak (eng Data-flow analysis) se uklapaju u okvir apstraktne interpretacije. Neke posebne analize koje su znacajne u deklarativnim jezicima:

Stroga analiza: Analiza koja omogucava otimizaciju lenjih funkcionalnih programa identifikujuci parameter koji mogu biti prosledjeni po vrednosti tako da se izbegne potreba za pravljenjem klozura (eng. closure) I otvara se mogucnost paralelne evaluacije.

Analiza menjanja u mestu: Ova analiza nam omogucava da odredimo tacke u programu na kojima je sigurno da se unisit objekat jer ni jedna pokazivac ne pokazuje na njega. Rezultate u ovoj oblasti je doneo Hudak. Znacajan rezultat je, po prvi put, funkcionalna verzija kviksort algoritma moze da se pokrene u linearnom prostoru. [4]

Analiza relevantnih klauza: U mnogim prototipovima 5 generacije arhitekture programi mogu da naprave ne-lokalni pristup definicjama funkcija. Ovo povlaci da postoji komunikacija povezan sa izvrsavanjem programa. Koriscenje analize delova postaje moguce identifikovati delove definicje funkcije koji su relevantni za nas program i tak smanjiti troskove.

Analiza moda: Znacajno povecanje preformansi moze se postici u Prologu ako zna kako se logicke varijable koriste ue relaciji (kao ulazne, izlazne ili mesavina ovo dvoje). Kada su deklarativna zajednica postala svesna apstraktne interpretacije, nove aplikacije su otkrivene. Optimizacije bazirane na apstraktoj interpretaciji su verovatno tacne. Ako ovo prebacimo u gornje primere to bi bilo:

Ко жели, може да пише рад ћирилицом. У том случају, неопходно је да су инсталирани одговарајући пакети: texlive-fonts-extra, texlive-latex-extra, texlive-lang-cyrillic, texlive-lang-other.

Uz sve novouvedene termine u zagradi naglasiti od koje engleske reči termin potiče. Naredni primeri ilustruju način uvođenja enlegskih termina kao i citiranje.

Primer 2.1 *Problem zaustavljanja (eng.* halting problem*) je neodlučiv [7].*

Primer 2.2 *Za prevođenje programa napisanih u programskom jeziku C može se koristiti GCC kompajler [3].*

Primer 2.3 *Da bi se ispitivala ispravost softvera, najpre je potrebno precizno definisati njegovo ponašanje [5].*

Reference koje se koriste u ovom tekstu zadate su u datoteci *seminarski.bib*. Prevođenje u pdf format u Linux okruženju može se uraditi na sledeći način:

```
pdflatex TemaImePrezime.tex
bibtex TemaImePrezime.aux
pdflatex TemaImePrezime.tex
pdflatex TemaImePrezime.tex
```

Prvo latexovanje je neophodno da bi se generisao *.aux* fajl. *bibtex* proizvodi odgovarajući *.bbl* fajl koji se koristi za generisanje literature. Potrebna su dva prolaza (dva puta pdflatex) da bi se reference ubacile u tekst (tj da ne bi ostali znakovi pitanja umesto referenci). Dodavanjem novih referenci potrebno je ponoviti ceo postupak.

Broj naslova i podnaslova je proizvoljan. Neophodni su samo Uvod i Zaključak. Na poglavlja unutar teksta referisati se po potrebi.

Ovde pišem uvodni tekst. Ovde pišem uvodni tekst. Ovde pišem uvodni tekst. Ovde pišem uvodni tekst.

3 Slike i tabele

Slike i tabele treba da budu u svom okruženju, sa odgovarajućim naslovima, obeležene labelom da koje omogućava referenciranje.

Primer 3.1 *Ovako se ubacuje slika. Obratiti pažnju da je dodato i*

```
\usepackage{graphicx}
```

Na svaku sliku neophodno je referisati se negde u tekstu. Na primer, na slici 1 prikazane su pande.

Primer 3.2 *I tabele treba da budu u svom okruženju, i na njih je neophodno referisati se u tekstu. Na primer, u tabeli 1 su prikazana različita poravnanja u tabelama.*

Tabela 1: Razlčita poravnanja u okviru iste tabele ne treba koristiti jer su nepregledna.

centralno poravnanje	levo poravnanje	desno poravnanje
a	b	c
d	e	f

4 Formalizacija

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

4.1 Prvi podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

4.2 Drugi podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

5 Primena

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

5.1 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

6 n-ti naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

6.1 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

6.2 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

7 Poslednji naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

8 Zaključak

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

Literatura

[1] S. Abramsky and C. Hankin. An introduction to abstract interpretation. pages 5–41, 1990.

[2] R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. MORGAN KAUFMANN PUBL Incorporated, 2001.

[3] Free Software Foundation. GNU gcc, 2013. on-line at: http://gcc.gnu.org/.

[4] J. Y. Girard and Y. Lafont. *Linear logic and lazy computation*, pages 52–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

[5] J. Laski and W. Stanley. *Software Verification and Analysis*. Springer-Verlag, London, 2009.

[6] Robert R. Schaller. Moore's Law: Past, Present, and Future. *IEEE Spectr.*, 34(6):52–59, June 1997.

[7] A. M. Turing. On Computable Numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe.