Sinteza programa

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

Anja Ivanišević, Ivan Ristović, Milana Kovačević, Vesna Katanić
kontakt email prvog, drugog (trećeg) autora

??. ?? 2018.

Abstract

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.

Contents

1 Uvod

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

Teorema 1.1 *Problem zaustavljanja (eng.* halting problem*) je neodlučiv [4].*

Teorema 1.2 *Za prevođenje programa napisanih u programskom jeziku C može se koristiti GCC kompajler [1].*

Teorema 1.3 *Da bi se ispitivala ispravost softvera, najpre je potrebno precizno definisati njegovo ponašanje [3].*

Reference koje se koriste u ovom tekstu zadate su u datoteci *literature.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.

Teorema 1.4 *U odeljku* **??** *precizirani su osnovni pojmovi, dok su zaključci dati u odeljku 6.*

Još jednom da napomenem da nema razloga da pišete:

```
\v{s} i \v{c} i \'c ...
```

Možete koristiti srpska slova

```
š i č i ć ...
```

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

2 Primene

The synthesized program may be explicitly presented to the user for debugging, re-use, or for being incorporated as part of a larger workflow. However, in some cases, the synthesized program may be implicit and is simply used to automate the intended one-off task for the user, as in case of spreadsheet string transformations [43].

3 Izazovi

Pisanje programa koji može da sintetiše drugi program predstavlja veliki izazov. Naime, ovaj problem se može razložiti na dva potproblema:

- Definisanje specifikacija željenog programa,

- Pretraživanje prostora mogućih programa u potrazi za onim koji zadovoljava definisane specifikacije.

Prostor programa se povećava eksponencijalno brzo u odnosu na veličinu željenog programa. Zbog toga postoje različiti pristupi njegovog pretraživanja, a neke od tih tehnika su opisane u poglavlju 3.2.

3.1 Definisanje specifikacija

Generisani program treba da se ponaša na način koji to korisnik definiše. Međutim, precizno definisanje zahteva je zapravo mnogo teže nego što izleda na prvi pogled. Postoje različiti načini na koje se to može program može opisati. Može se opisati formalnim logičkim izrazima, kao i neformalnim metodama ili primerima ulaza i izlaza programa.

Formalno definisanje zahteva može često da izgleda komplikovano (možda čak i da deluje komplikovanije neko pisanje samog programa). Nasuprot tome, neformalne metode su mnogo prirodnije korisniku, ali dovode do drugih problema. Na primer, neka se željeni program definiše na osnovu primera njegovog ulaza i izlaza na sledeći način: *"John Smith" -¿ "Smith, J."*. Ovaj program vrši nama intuitivnu transformaciju niski, ali, na primer, da bi se on automatski generisao korišćenjem FlashFill [2] programa, tj. program treba da pretraži prostor koji sadrži milione mogućih rešenja. Problem je u tome što programi nemaju ljudsku intuiciju, već se preprilagođavaju datim primerima ulaza i izlaza.

Većina programa koji se danas koriste su previše komplikovani da bi se u potpunosti opisali bilo formalnim bilo neformalnim metodama. Čak i ako bi se to nekako uspelo, opis programa bi mogao da bude toliko obiman kao i sama implementacija programa. Kako bi senteza ovakvih, realnih programa bila moguća, potrebno je omogućiti korisniku da na početku definiše željeni program do neke tačke, a da kasnije tokom sinteze, interaktivno sa računarom, postepeno dolazi do rešenja.

Upravo ovakvu napradnu pretragu koristi gore spomenuti program FlashFill. Tokom pretrage, on uključuje dodatnu komunikaciju sa korisnikom. Ovako on usmerava pretragu, te na kraju ipak uspeva da nađe rešenje u realnom vremenu.

3.2 Prostor programa

Svaka uspešna sinteza programa vrši neki vid pretrage prostora mogućih programa (eng. *search space*). Ovo je težak kombinatorni problem. Broj mogućih rešenja raste eksponencijalno sa veličinom programa, te pretraga svih kandidata nije moguća u realnom vremenu. Potrebno je pažljivo vršiti odsecanja dela prostora pretrage kako bi se došlo do rešenja u realnom vremenu.

Tehnike pretrage se mogu zasnivati na enumerativnoj pretrazi, dedukciji, tehnikama sa ograničenjima, statističkim tehnikama, kao i na kombinaciji nekih od nih.

3.2.1 Enumerativna pretraga

Tehnike enumerativne pretrage za sintezu prigrana su se pokazale kao jedne od najefikasnijih tehnika za generisanje malih programa. Razlog ove efikasnosti je u pametnim tehnikama *čišćenja* (eng. *pruning*) u prostoru programa koji se pretražuje. Glavna ideja je da se prvo na neki način opiše prostor pretrage u kome se nalazi žejleni program. To može da

se postigne korišćenjem meta-podataka kao što su veličina programa ili njegova složenost. Kada se mogući programi numerišu po osobinama, mogu da se odmah odbace oni koji ne zadovoljavaju prethodno definisane specifikacije.

Kako na osnovu pretpostavki vrši velika odsecanja, može da se dođe do toga da pogrešno numeriše neki od programa i time izgubi neka od mogućih rešenja. Zato je enumerativna tehnika polu-odlučiva, ali u opštem slučaju je upotrebljiva i daje dobre rezultate i to relativno brzo.

3.2.2 Deduktivna pretraga

Deduktivna sinteza programa je tradicionalni pogled na sintezu programa. Ovakvi pristupi pretpostavljaju da postoji celokupna formalna specifikacija željenog programa. Ovo je vrlo jaka pretpostavka imajući u vidu da ta specifikacija može da bude veoma velika ukoliko je program kompleksan. Rešenje se sintetiše postupkom dokazivanja teorema, logičkim zaključivanjem i razrešavanjem ograničenja.

Deduktivna pretraga je pretraga odozgo - na dole. Koristi tehniku podeli-pa-vladaj (eng. *divide-and-conquer*). Program sintetiše tako što se prvo podeli na potprobleme tako da svaki od njih ima svoju specifikaciju. Rekurzivno se obrade potproblemi, a zatim iskombinuju podrešenja kako bi se dobilo glavno rešenje.

Deljenje problema na potprobleme koji mogu da se sintetišu odvojeno nije moguće u opštem slučaju. Ovo zavisi od prirode problema. U tom slučaju se deduktivna pretraga može iskombinovati sa enumerativnom. Kada deduktivna pretraga više ne može da razloži problem, enumerativnom pretragom (koja je odozdo - na gore) tda treba pretražiti prostor rešenja potproblema, i nakon toga spojiti dobijene rezultate.

If the underlying grammar allows for a rich set of constants, the bottom-up enumerative search can get lost in simply guessing the right constants. On the other hand, the top-down deductive technique can deduce constants based on the accumulated constraints as the last step in the search process.

3.2.3 Tehnike sa ograničenjima

Mnoge uspešne tehnike sinteze programa u svojoj osnovi sadrže tehnike prilagođavanja datim ograničenjima (eng. *constraint solving*). One se sastoje od dva velika koraka:

- *Generisanje ograničenja* - U opštem slučaju, kada se kaže prilagođavanje ograničenjima misli se na na pronalaženje modela za formulu koja opisuje željeni program. Osnovna ideja je da se specifikacija programa kao i njegova dodatna ograničenja zapišu u jednoj logičkoj formuli. Uglavnom se tom prilikom u formulu dodaju pretpostavke o rešenju.
- *Razrešavanje ograničenja* - Formula u kojoj su zapisana ograničenja često sadrži kvatifikatore i nepoznate drugog reda. Ova formula se prvo transformiše u oblik pogodan za nekog od rešavača, na primer SAT ili SMT rešavač. Na ovaj način se problem pretrage svodi na problem ispitivanja zadovoljivosti prosleđene formule. Svaki nađeni model za tu formulu predstavlja jedno moguće rešenje koje zadovoljava data ograničenja.

TODO: primer? We illustrate this idea on a simple example. Consider a small DSL of bitwise operations upon a 8-bit input variable x: ...

3.2.4 Statistička pretraga

Postoji veliki broj statističkih metoda koje mogu da se upotrebe za pretragu. Neke od njih su:

- *Mašinskog učenje* - Tehnike mašinskog učenja mogu doprineti ostalim pretragama uvodeći verovatnoću u čvorove granjanja prilikom pretrage. Vrednosti verovatnoće se uglavnom generišu pre sinteze programa: tokom treninga ili na primer na osnovu datih primera ulaza i izlaza.
- *Genetičkog programiranje* - Genetičko programiranje je metod inspirisan biološtkom evolucijom. Sastoji se od održavanja populacije programa, njihovog ukrštanja i mogučih mutacija. Svaka jedinka populacije se ispituje u kojoj meri zadovoljava specifikacije žejlenog programa. One jedinke koje bolje odgovaraju rešenju nastavljaju da evoluiraju. Uspeh genetičkog programiranja zavisi od funkcije zadovoljivosti.
- *MCMC sampling* - MCMC sampling has been used to search for a desired program starting from a given candidate. The success crucially depends on defining a smooth cost metric for Boolean constraints. STOKE [124], a superoptimization tool, uses Hamming distance to measure closeness of generated bit-values to the target on a representative test input set, and rewards generation of (almost) correct values in incorrect locations.
- *Probabilistic inference* - Probabilistic inference has been used to evolve a given program by making local changes, one at a time. This relies on modeling a program as a graph of instructions and states, connected by constraint nodes. Each constraint node establishes the semantics of some instruction by relating the instruction with the state immediately before the instruction and the state immediately after the instruction [45]. Belief propagation has been used to synthesize imperative program fragments that execute polynomial computations and list manipulations [62].

4 CEGIS

What is CEGIS? Synthesis tasks often have the same structure: an implementation is sought that behaves correctly under all possible inputs (with the help of some extra variables, i.e. helper variables).

It is absolutely legal to pass such a term to an SMT solver like Z3. A big drawback is the universal quantifier though.

For many real world problems it is not necessary to consider all inputs to derive an implementation that behaves correctly for all of them. Following this observation, the problem was just moved to another position: which is the minimal subset of inputs one have to consider to ensure a correct synthesis?

This is the point where CEGIS comes into play. CEGIS is a loop looking for exactly this minimal subset of inputs and performing the implementation synthesis as a "by-product". Therefore CEGIS uses one satisfiability solver to generate new implementations based on all the inputs considered so far (starting with zero); and another one to generate counter examples that uncover incorrect behavior in the latest synthesised

implementation. Eventually there will be no more implementations possible, i.e. the specification is not realisable, or no more counter examples possible, i.e. the latest implementation must be correct.

This CEGIS library works with the SMT solver Z3 and requires insight in the synthesis task to be executed, as it has to be specified which variables belong to implementation, inputs, etc. Boundary conditions are to be specified manually as well.

5 Slike i tabele

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

Teorema 5.1 *Ovako se ubacuje slika. Obratiti pažnju da je dodato i* `\usepackage{graphicx}`

Figure 1: Pande

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

Teorema 5.2 *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.*

Table 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

6 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.

References

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

[2] Sumit Gulwani. *Automating string processing in spreadsheets using input-output examples*. Symposium on Principles of Programming Languages, POPL, Austin, TX, USA, 2011.

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

[4] 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.