

# Lokalizacija grešaka korišćenjem Maksimalne zadovoljivosti

Seminarski rad u okviru kursa  
Verifikacija softvera  
Matematički fakultet

Filip Novović  
filnovovic@gmail.com

4. maj 2018.

## Sažetak

U procesu izrade softvera, veliki deo vremena se potroši na lokalizaciju delova koda koji mogu izazvati greške pri izvršavanju. U ovom radu je opisan algoritam [3] koji se može koristiti za pronalaženje tih delova koda, a svodi se na problem maksimalne zadovoljivosti (eng. maximal satisfiability problem MAX-SAT). Rešenje problema maksimalne zadovoljivosti predstavlja najveći broj formula iz ulaznog skupa koji se može zadovoljiti nekom valuacijom.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Prikaz primera</b>	<b>2</b>
2.1	Interpretacija koda u BugAssist-u . . . . .	3
2.2	Ispravka programa . . . . .	3
<b>3</b>	<b>Algoritam</b>	<b>4</b>
3.1	Generisanje tragova . . . . .	4
3.2	Lokalizacija . . . . .	4
<b>4</b>	<b>Zaključak</b>	<b>4</b>
	<b>Literatura</b>	<b>5</b>

# 1 Uvod

**Ulaz** u algoritam za lokalizaciju linija koda koje mogu biti uzrok grešaka jesu program, specifikacija tačnosti (post-uslov koji mora biti zadovoljen ili pretpostavka) i ulaz za zadati program koji narušava specifikaciju (pri njegovoj obradi dolazi do greške pri izvršavanju programa).

**Izlaz** iz algoritma je najmanji skup programskih naredbi za koje postoji adekvatna izmena tako da program za zadati ulaz uspeva da se izvršava bez grešaka.

**Koraci** algoritma su:

1. Konstrukcija simboličke formule traga (eng. trace formula) koja predstavlja putanju programa pri izvršavanju zadatog ulaza. Ona je iskazna formula u KNF-u i zadovoljiva je ako i samo ako je izvršavanje programa izvodljivo.
2. Formula traga proširuje se tako što joj pridružujemo ograničenja koja obezbeđuju da ulazno stanje zadovoljava vrednosti zadatog ulaza, a završna stanja zadovoljavaju post-uslove. Primećujemo da je proširena formula traga za zadati ulaz nezadovoljiva.
3. Proširena formula traga se prosleđuje rešavaču maksimalne zadovoljivosti (eng. maximum satisfiability solver). Rešavač iz skupa formula proširene formule traga pronalazi najveći skup formula koje se mogu zadovoljiti, taj skup zatim komplementiramo i dobijamo kandidatski skup klauza koji se može izmeniti tako da cela formula postane zadovoljiva. Svaka formula proširene formule traga se preslikava na neku naredbu iz ulaznog programa pa one ujedno predstavljaju kandidatsku lokalizaciju greške izraženu samim naredbama.

**Rešavač parcijalnog MAX-SAT problema** . Algoritam koji opisujemo koristiće parcijalni MAX-SAT rešavač kod kog se ulazne klauze mogu klasifikovati kao *tvrde* (nisu dozvoljene njihove promene) i *meke* (dozvoljene su promene). Parcijalni rešavač se razlikuje po tome što maksimalni skup klauza koje se mogu zadovoljiti traži u mekim klauzama (tvrde se moraju zadovoljiti). Kod algoritma koji opisujemo, ulazni uslovi i post-uslovi su tvrde klauze što je neophodno jer bi u suprotnom rešavač mogao da izmeni željeni rad programa ili ulaz.

**BugAssist** je program koji implementira algoritam koji se obrađuje u ovom radu i podržava programski jezik C. Njegov ulaz je C program sa pretpostavkama i skupom testnih slučajeva pri čijem izvršavanju dolazi do greške.

## 2 Prikaz primera

Deo koda koji analiziramo

```
1 int Array[3];
2 int testme(int index){
3     ...
4     if(index != 1) // Potencijalni problem 2
5         intex=2;
6     else
```

```

7       index=index+2; // Potencijalni problem 1
8       ...
9       i=index;
10      return Array[i]; //assert(i>=0 && i<3)
11  }

```

Funkcija testme vraća vrednost sa nove lokacije iz niza dužine 3. Ulazni parametri ove funkcije su niz elemenata i trenutni indeks. Izračunava se novi indeks na osnovu trenutnog i vraća se vrednost koja odgovara poziciji tog novog indeksa. Lako se primećuje da ovaj kod sadrži greške. Ukoliko je ulazna vrednost promenljive index jednaka 1, vrednost novog indeksa će biti 3 pa će linija koda koja pristupa elementu na toj poziciji (linija 10) izazvati grešku zato što pristupamo elementu koji je van niza.

## 2.1 Interpretacija koda u BugAssist-u

Za testni ulaz  $index=1$ , odgovarajući trag programa je:

```
assume(index=1);index=index+2;i=index;
```

na osnovu nje on konstruiše simboličku formulu traga TF:

$$TF \equiv index_1 \wedge index_2 = index_1 + 2 \wedge i = index_2$$

Uvodimo novu formulu:

$$\Phi \equiv index_1 = 1 \wedge TF \wedge i < 3$$

U navedenoj formuli  $\Phi$  leva strana u odnosu na TF odgovara testnom ulazu, a desna pretpostavci. Ova formula nije zadovoljiva.

Formulu  $\Phi$  zapisujemo u KNF formi i zadajemo je kao ulaz parcijalnom MAX-SAT rešavaču. Tvrde klauze u ovom slučaju će nam biti testni ulaz ( $index=1$ ) i pretpostavka ( $i<3$ ), a ostale će biti meke. Parcijalni MAX-SAT rešavač pokušava da nađe skup mekih klauza najveće kardinalnosti koje mogu istovremeno da zadovolje sve tvrde klauze. Komplement skupa maksimalno zadovoljivih klauza (eng. Complement of a set of maximum satisfiability clauses - CoMSS) vraća skup mekih klauza minimalne kardinalnosti čije uklanjanje bi učinilo da  $\Phi$  bude zadovoljiva. MAX-SAT vraća da CoMSS ukazuje na liniju 4 u kodu. Ona predstavlja osnovu nezadovoljivosti čije izbacivanje ili ispravka mogu dovesti do zadovoljivosti formule  $\Phi$ .

Ukoliko ovo nije linija na kojoj programer želi da napravi izmenu i želi neku drugu lokaciju za ispravku, on može klauzu koja odgovara toj liniji proglasiti tvrdom pa napraviti nov poziv MAX-SAT rešavaču. Ovaj proces se ponavlja sve dok postoje klauze koje se mogu izmeniti tako da cela formula postane zadovoljiva.

## 2.2 Ispravka programa

Metodologija za lokalizaciju grešaka može se preraditi kako bi dobila mogućnost da programu predlaže ispravke. Intuitivno, pronalazak grešaka vraća skup programskih naredbi za koje postoji velika verovatnoća da su pogrešne. Generalno prostor potencijalnih izmena ovih naredbi je jako veliki i njegova efikasna pretraga predstavlja težak problem. Umesto toga, možemo obratiti pažnju na česte programerske greške.

Fokusiraćemo se na greške tipa "pomerenost za jedan" (eng. off by one).

One se manifestuju tako što do greške dolazi pri pokušaju da se pristupi elementu koji je van granice niza za jedan indeks. Kada BugAssist vrati liniju 4 kao potencijalnu lokaciju greške, mi pokušavamo da je popravimo tako što menjamo konstante za jednu vrednost naviše ili naniže.

### 3 Algoritam

Postoje dve faze algoritma. U prvoj se izazivaju greške i generišu testni slučajevi koji im odgovaraju. U drugoj se traži minimalni skup izmena koje obezbeđuju da se greške uklone.

#### 3.1 Generisanje tragova

Uglavnom se može koristiti bilo koji metod za generisanje testnih ulaza za koje postoje greške i koje se mogu koristiti kao početna tačka algoritma. U ovom primeru se koriste dva pristupa. U slučaju da program dobijamo sa nizom testova (eng. testsuite), generišemo problematična izvršavanja (eng. failing executions) na osnovu testova koji izazivaju greške. Ako nemamo moguće testove, koristimo proveru ograničenog modela (eng. bounded model checking) [1][2] kako bismo sistemski istražili izvršavanje programa i našli potencijalna narušavanja pretpostavki. Ako se naide na problematično izvršavanje možemo generisati konkretno inicijalno stanje koje vodi do narušavanja pretpostavki.

#### 3.2 Lokalizacija

Na slici 1 možemo videti algoritam koji se koristi za lokalizaciju grešaka.

```

Input: Program  $\mathcal{P}$  and assertion  $p$ 
Output: Either  $p$  holds for all executions or potential bug locations
1:  $(\text{test}, \sigma) = \text{GenerateCounterexample}(\mathcal{P}, p)$ 
2: if  $\sigma$  is "None" then
3:   return "No counterexample to  $p$  found"
4: else
5:    $\Phi_H = [\text{test}] \wedge p \wedge \text{TF}_1(\sigma)$ 
6:    $\Phi_S = \text{TF}_2(\sigma)$ 
7:   while true do
8:      $\text{BugLoc} = \text{CoMSS}(\Phi_H, \Phi_S)$ 
9:     if  $\text{BugLoc} = \emptyset$  then
10:      return "No more suspects"
11:    else
12:      output "Potential bug at CoMSS BugLoc"
13:       $\beta = \bigvee \{\lambda_i \mid \lambda_i \in \text{BugLoc}\}$ 
14:       $\Phi_S = \Phi_S \setminus \beta$  and  $\Phi_H = \Phi_H \cup \beta$ 

```

Slika 1: Algoritam lokalizacije

### 4 Zaključak

Analiza programa zasnovana na zadovoljivosti iskaznih formula se pokazala vrlo uspešnom u otkrivanju čak i manjih grešaka u velikim programima. Opisana tehnika koristi moderne SAT i MAX-SAT rešavače i pred-

stavlja precizno i skalabilno rešenje problemu lokalizacije grešaka. Zanimljive teme u budućnosti bi mogle da budu razne automatizacije ispravki grešaka u kodu analizom njihovih lokacija što zahteva predviđanje tipova grešaka koje imaju najveću verovatnoću da se pojave u određenom izrazu.

## Literatura

- [1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, DAC '99*, pages 317–320, New York, NY, USA, 1999. ACM.
- [2] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [3] Manu Jose and Rupak Majumdar. Cause clue clauses: Error localization using maximum satisfiability. *SIGPLAN Not.*, 46(6):437–446, June 2011.