

Efikasno nalaženje bagova pomoću SAT rešavača

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Filip Miljaković
filipmiljakovic1994@gmail.com

7. maj 2018.

Sažetak

Predstaviću vam pristup koji se zasniva na kodiranju programa u relacionoj logici, koristeći ograničen rešavač za nalaženje stvari koje se razlikuju od specifikacije. Ovaj pristup mnogo bolje deli program na logičke celine, zahvaljujući novom kodiranju. Takođe biće predstavljeno novo kodiranje nizova i celih brojeva, koje radi sa većim opsezima nego što je to pre bio slučaj i sa velikim, retkim, nizovima. Ova tehnika je implementirana i proverena nad invarijantama struktura podataka nekoliko klasa iz Java Collection Framework-a. Takođe proveravano je i tumačenje ekvivalenosti u više opensource programa.

1 Uvod

[1] Naučnici su razvili širok spektar automatizovanih tehnika da pronađu bagove u kodu, uključujući testiranje i statičke analize.

Pristup sa testiranjem podrazumeva podskup svih mogućih ponašanja programa i ima tu prednost ostavljanja konkretnih dokaza postojanja tih bagova. Međutim, pristup sa testiranjem može da sadrži propuste zbog nedovoljne pokrivenosti koda testovima.

Tehnika statičke analize aproksimiraju sva ponašanja programa. Međutim, statička analiza može generisati netačne izveštaje o grešci zbog nepreciznosti.

U ovom radu fokus će biti na proveru ekspresivnih osobina manipulacije sa heap-om u objektu orijentisanom kodu koristeći relaciju logiku i rešavanjem ograničenja.

Mali skup postavljenih hipoteza smatra da mali heap(samo nekoliko objekata) može efikasno da se pokrije testovima koji testiraju manipulacije nad njim.

Na primer, za proceduru koja sortira povezanu listu, verovatno je dovoljno testirati liste malih veličina pa ostala testiranja postaju suvišna.

Postojeći relacioni pristup kodira program u relacionu formulu prvog reda koja predstavlja sva izvršavanja programa ograničena na male konačne petlje. Ovo postavlja ograničenje na broj objekata po svakom tipu. Korisnik dalje daje specifikaciju u logici prvog reda.

Moderni SAT rešavači mogu da nađu kontraprimere efikasno u velikim formulama. Benefit ovog pristupa je modularnost.

Tradicionalan proverivači modela moraju da navedu inicijalni sadržaj sa kojim je kod iscrpno testiran, a relacione tehnike mogu pretražiti sve moguće kontekste i na taj način mogu otkriti suptilne, neočekivane greške. Prethodni relacioni pristup ima dva nedostatka:

Prvo, kod i specifikacija su prevedeni odvojeno, tako da kodiranje ne može imati koristi ako se specifikacija odnosi samo na male delove koda.

Drugi, celi brojevi su tretirani kao objekti, predstavljajući eksplicitno svaki ceo broj. Prema tome, opseg brojeva za analizu je strogo ograničen i veliki brojevi(konstante ili indeksi nizova)su suviše skupi za analizu.

Ove nedostatke rešavamo na sledeći način:

Prvo, prevođenja iz koda u relacionu logiku prvog reda vrši tip sečenja na logičkom nivou, zasnovanom na promenljivama i instanciranim poljima koji se nalaze u specifikaciji i samo relevantni delovi programa su kodirani.

Drugo, kodiramo cele brojeve implicitno ali precizno, kao sumu stepena dvojke. Ovo nam dozvoljava da predstavimo cele brojeve iz mnogo većeg opsega nego ranije. Ovaj alat može da obradi 16-24-bitne cele brojeve, dok je prethodni mogao 4 bita.

Treće, ovde kodiramo nizove tako da mogu biti indeksirani velikim brojevima.

Napravljen je Miniatur, implementacija ove tehnike, i korišćen je na dva skupa programa za testiranje.

Prvi se ticao proveravanja invarijanti Java kolekcijских klasa, uključujući HeshMap-e i drveta, što demonstrira njegovu sposobnost da radi sa velikim brojevima i nizovima.

Drugi deo uključuje proveru java jednakosti, za razne java opensource programe.

2 Motivacija

Miniatur uzima kao ulaz procedure i parametre koda kao što su: broj iteracija petlje, broj bitova kod celih brojeva i broj objekata kod inicijalnog heap-a. Korisnik može ukazati na specifikaciju koristeći Java tvrđenja (eng. *assertion*), ili specijalna tvrđenja koja sadrže relacione formule. Ako postoji neko tvrđenje koja ne važi, Miniatur-ov izlaz je inicijalni heap i parametri.

Java dodeljuje svakom objektu jedinstven identitet, ali i dozvoljava programeru da prilagođava identitet predefinisanjem metoda **equals()** i **hashCode()**, prateći pravila, koja zahtevaju, između ostalog, da relacija jednakosti bude refleksivna, simetrična i tranzitivna i da jednaki objekti imaju jednake hash kodove. Međutim dogovori su skloni greškama. Java API za rad sa kolekcijama zahteva da tipovi objekata koji se nalaze u kolekcijama budu isti do na jednakost i da u suprotnom imaju neočekivano ponašanje.

Naša tehnika se može koristiti za proveru npr. definisanja jednakosti u Javi, i u izradi konkretnih kontraprimera. Generisanje velikih brojeva je vitalno za svaku dobru hash funkciju. Miniatur-ova podrška za velike cele brojeve je ključni doprinos naše tehnike.

3 Pozadina implementacije

Mi kodiramo ponašanje procedure u logičku formulu koja je zadovoljiva, ako postoji izvršavanje koje je u suprotnosti sa specifikacijom. Onda koristimo pronalazač modela(zasnovan na SAT rešavaču) za nalaženje rešenja.

3.1 Pozadina rešenja

Koristimo relacionu logiku prvog reda. Njena gramatika je data na slici 1. Osnovni koncept je relacija: skup torki jednake veličine. Arnost relacije je broj elemenata u svakoj torci, a njihova veza je domen odakle torke uzimaju svoje vrednosti. Relaciona logika omogućava razne operacije nad relacijama. Relaciona logika takođe manipuliše bitovima.

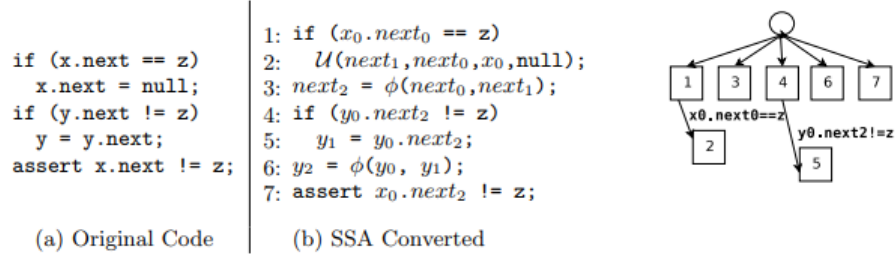
$$\begin{aligned} \text{Form} ::= & \text{Expr } \{=, \neq, \subset, \supset\} \text{ Expr} \mid \\ & \text{Form } \{\wedge, \vee, \Rightarrow\} \text{ Form} \mid \neg \text{Form} \mid \\ & \{\forall, \exists\} x : \text{Expr} . \text{Form} \mid \{\text{no}, \text{one}\} \text{Expr} \mid \\ & \text{Bitset } \{=, \neq, <, >, \leq, \geq\} \text{ Bitset} \\ \\ \text{Expr} ::= & r \mid \text{Expr } \{., -, >, ++, +, -\} \text{ Expr} \mid \{x : \text{Expr} \mid \text{Form}\} \\ & \mid \text{if Expr then Expr else Expr} \mid \text{Expr}^* \mid |\text{Expr}| \\ \\ \text{Bitset} ::= & v \mid \text{Bitset } \{+, -, *, /, \wedge_b, \vee_b\} \text{ Bitset} \mid \text{sum Expr} \end{aligned}$$

Slika 1: Relaciona logika - Sintaksa

3.2 SSA forma i CDG

Ovo kodiranje koristi statičku pojedinačnu podelu forme i graf kontrole zavisnosti da efikasno kodira informacije o zavisnostima podataka i

kontrole nad njima. Primenjuje se SSA forma na polja isto kao i na podatke. CDG ima polja kao čvorove i ivice od uslovnih izraza do izraza koji direktno kontrolišu zavisnosti nad njima. Svaka ivica je označena uslovom koji uslovljava izvršavanje njenog ciljanog izraza.



Slika 2: Primer SSA i CDG

Imamo primer SSA forme i CDG za ovaj primer na slici 2.

4 Prevođenje

4.1 Osnovno kodiranje

Osnovno kodiranje radi na sledeći način:

Prvo prevedemo kod u SSA formu i napravimo CDG. Oslanjamo se na translacionu funkciju T koja uzima izraz i vraća relacioni izraz u smislu inicijalnog stanja.

4.2 Funkcija translacije T

To je parcijalna funkcija koja uzima promenljivu ili polje(u SSA formi) i vraća izraz u kodu koje definiše njenu vrednost. Za promenljivu ili polje x , $T(x)$ vraća relacioni izraz koji daje vrednost x u smislu inicijalnog stanja. T prevodi, dereferencira polja koristeći relacioni operator unije.

5 Evaluacija

Paš prototip Miniatur-a može analizirati veći deo Java jezika. Njegov unos se sastoji od procedura koje sadrže neka tvrđenja, kao i od parametara npr.: broj iteracija petlje, sirina bitova kod celih brojeva, broj objekata u inicijalnom heap-u, broj atoma korišćenih za indeksiranje nizova. Izlaz našeg programa je inicijalni heap i parametri, sa kojima dolazi do kršenja neke pretpostavke (eng. *assertion*), ako takva postoji.

Miniatur određuje tipove potrebne za analizu i ograničenja nad ovim tipovima na sledeći način. Da bismo minimizovali broj tipova, generišemo samo one koje će program možda trebati.

Tu spadaju 4 vrste tipova:

Oni koji se eksplicitno imenuju u operacijama za proveru tipova (kastovanja, instanceof operacijama)

Tipovi korišćeni u dinamičkim dispatch operacijama, koji su potencijalno primaoci virtualnih poziva.

Tipovi koji se koriste za čitanje i pisanje.

Tipovi argumenata i onih koji se tranzitivno mogu dobiti od tih argumenata, korišćenjem pokazivača.

Naravno, postoji mnoštvo primera kojima se mogu bliže pokazati sve mogućnosti ovog alata i koji se mogu naći, ali ih zbog nedostatka mesta ovde neću prikazivati.

6 Zaključak

Prikazali smo metodu za pronalaženje bugova u kodu na temelju relacije logike i SAT rešavača, što pruža puno bolju skalabilnost. Može proveravati mešavinu strukturnih i numeričkih osobina napisanih u bogatoj specifikaciji jezika sa realnim fragmentima programa. Koliko se zna, ovaj alat je prvi potpuno automatizovani alat koji proverava kršenje dogovora o jednakosti u stvarnim programima. U budućnost, planira se proširiti ova tehniku kako bi se uticalo na drugačije analize programa kako bi se smanjio prostor koji SAT rešavač pretražuje, i pokušalo poboljšati izvođenje aritmetičkih operacija.

Literatura

- [1] Mandana Vaziri Julian Dolby and Frank Tip. Finding bugs efficiently with a sat solver, jan 2007.