UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Ana Đorđević

AUTOMATSKO GENERISANJE TEST PRIMERA UZ POMOĆ STATIČKE ANALIZE I REŠAVAČA Z3

master rad

Beograd, 2017.

Mentor:

dr Milena Vujošević Janičić, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Filip Marić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Milan Banković, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami i tati

Naslov master rada: Automatsko generisanje test primera uz pomoć statičke analize i rešavača Z3

Rezime:

Ključne reči: verifikacija softvera, testiranje softvera, SMT rešavači, Z3 rešavač, računarstvo

Sadržaj

Glava 1

Uvod

Glava 2

Razrada

2.1 Rešavač Z3

Uopšteno o rešavaču

Sistemi za analizu i verifikaciju softvera su veoma kompleksni. Njihovu osnovu predstavlja komponenta koja koristi logičke formule za opisivanja stanja i transformacija između stanja sistema. Zadovoljivost u modularnoj teoriji (Satisfiability Modulo Theories) može biti korišćena za opisivanje stanja sistema, proveravanjem zadovoljivosti formula logike prvog reda. Rešavač Z3 je napredni SMT dokazivač kompanije Microsoft koji se može koristiti za proveru zadovoljivosti logičkih formula u velikom broju teorija. Neke od njih su teorija linearne aritmetike, teorija bit-vektora, teorija neiterpretiranih funkcija i teorija nizova i one će biti korišćene tokom rada. Z3 se najčešće koristi kao podrška drugim alatima, pre svega alatima za analizu i verifikaciju softvera. Rešavač Z3 nudi interfejs za direktnu komunikaciju sa programskim jezicima C, C++ i Python. Pripada grupi SMT rešavača sa integrisanim procedurama odlučivanja.

Ulazni format rešavača Z3 je definisan SMT-LIB 2.0 standardom. Interno, Z3 održava stek korisnički definisanih formula i deklaracija. Nazivamo ih tvrđenjima definisanih od strane korisnika. Komandom push kreira se novi opseg i čuva se trenutna veličina steka. Komandom pop uklanjaju se sva tvrđenja i deklaracije zadate posle push-a sa kojim se komanda uparuje. Komandom assert dodaje se formula na interni stek. Skup formula na steku je zadovoljiv ako postoji interpretacija u kojoj sve formule imaju istinitosnu vrednost tačno. Ova provera se vrši komandom

check-sat. U slučaju zadovoljivosti vraća se sat, u slučaju nezadovoljivosti vraća se unsat a kada rešavač ne može da proceni da li je formula zadovoljiva ili ne vraća se unknown. Komandom get-model vraća se interpretacija u kojoj su sve formule na steku tačne.

Glavni gradivni blokovi formula su konstante, funkcije i relacije. Konstante su specijalan slučaj funkcija koje nemaju parametre. Z3 ne vrši automatsku konverziju celobrojnih konstanti u realne i obrnuto. Funkcija to-real može se iskoristiti za konvertovanje celobrojnih vrednosti u realne. Relacije su funkcije koje vraćaju povratnu vrednost tipa Boolean. Funkcije mogu uzimati argumente tipa Boolean pa se na taj način relacije mogu koristiti kao argumenti funkcija. Za razliku od programskih jezika, gde funkcije imaju bočne efekte funkcije logike prvog reda su totalne, tj. definisane su za sve vrednosti ulaznih parametara. Na primer, deljenje 0 je dozvoljeno, ali nije specifikovano šta ono predstavlja. Funkcije i konstantni simboli su neiterpretirani. Ovo je kontrast u odnosu na funkcije odgovarajućih teorija. Funkcija + ima standardnu iterpretaciju u teoriji aritmetike. Neiterpretirane funkcije i konstante su maksimalno fleksibilne i dozvoljavaju bilo koju interpretaciju koja je u skladu sa ograničenjima. Sledi primer sa korišćenjem neinterpretiranih tipova i funkcija korišćenjem C++ API-ja **??**.

```
/*
   Funckija demonstrira upotrebu neiterpretiranih tipova
   i funkcija
   dokazivanjem jednakosti:
   x = y => g(x) = g(y)
*/
void prove_example1() {

    context c;
    expr x      = c.int_const("x");
    expr y      = c.int_const("y");
    sort I      = c.int_sort();
    func_decl g = function("g", I, I);
```

```
    solver s(c);
    expr conjecture1 = implies(x == y, g(x) == g(y));
    s.add(!conjecture1);
    if (s.check() == unsat)
        std::cout << "proved" << "\n";
    else
        std::cout << "failed to prove" << "\n";
}
```

Komandom declare-const deklariše se konstanta odgovarajuće sorte (odgovara tipu promenljive u programskim jezicima). Sorta može biti parametrizovana i u tom slučaju su specifikovana imena njenih parametara. Specifikacija sorte vrši se naredbom (define-sort [symbol] ([symbol]+) [sort]). Komandom declare-fun deklariše se funkcija. U narednom primeru, deklarišemo funkciju koja ima dva parametra tipa Integer i Boolean i vraća vrednost tipa Integer. Komandom assert, kao što je već navedeno, dodaje se odgovarajuće ograničenje a zatim se proverava zadovoljivost formule.

Primer 1 *Definisanje konstanti i funkcija*
(declare-const a Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)

Formula F je validna ako je vrednost valuacije true za bilo koje interpretacije funkcija i konstantnih simbola. Formula F je zadovoljiva ukoliko postoji evaluacija u kojoj je njena vrednost tačna. Da bismo odredili da li je formula F valjana, rešavač Z3 proverava da li je formula not F zadovoljiva. Ukoliko je negacija formule nezadovoljiva, onda je polazna formula validna. Dakle, da bismo dokazali De Morganov zakon, pokazujemo da je njegova negacija nezadovoljiva.

Primer 2 *Dokazivanje De Morganovog zakona*
(declare-const a Bool)

(declare-const b Bool)
(define-fun demorgan () Bool
(= (and a b) (not (or (not a) (not b)))))
(assert (not demorgan))
(check-sat)

Prethodni primer ilustruje se korišćenjem C++ **??** API-ja.

```
/**

   Dokazivanje De Morganovog zakona dualnosti: {e not(x and y) <-> (not x
*/
void demorgan() {


    context c;

    expr x = c.bool_const("x");
    expr y = c.bool_const("y");
    expr conjecture = !(x && y) == (!x || !y);

    solver s(c);
    // dodavanje negacije konjunkcije kao ogranicenja
    s.add(!conjecture);
    std::cout << s << "\n";
    switch (s.check()) {
    case unsat:   std::cout << "de-Morgan is valid\n"; break;
    case sat:     std::cout << "de-Morgan is not valid\n"; break;
    case unknown: std::cout << "unknown\n"; break;
    }
}
```

Aritmetike

Rešavač Z3 sadrži procedure odlučivanja za linearnu aritmetiku nad celobrojnim i realnim brojevima.
U realnoj linearnoj aritmetici, interpretirani funkcijski simboli su +, ,- ,* , i (unarna negacija). Konstante se mogu porediti korišćenjem operatora =, <, <=, >, >=. U celobrojnoj linearnoj aritmetici, interpretirani funkcijski simboli su +, -, *, , mod i div. U slučaju funkcija mod i div drugi argument mora biti konstanta različita od 0.
Z3 ima podršku za celobrojne i realne konstante. Komandom declare-const deklarišu se celobrojne i realne konstante.

Primer 3 *Deklarisanje celobrojnih i realnih konstanti*

(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Real)
(declare-const e Real)

Takođe, postoji uslovni operator (if-then-else operator). Na primer, izraz (ite (and (= x!1 11) (= x!2 false)) 21 0) ima vrednost 21 kada je promenljiva x!1 jednaka 11, a promenljiva x!2 ima vrednost False. U suprotnom, vraća se 0.

U slučaju deljenja, može se koristiti ite (if-then-else) operator i na taj način se može dodeliti interpretacija u slučaju deljenja nulom.
Mogu se konstruisati novi operatori, korišćenjem define-fun konstruktora. Ovo je zapravo makro, pa će rešavač vršiti odgovarajuće zamene.

Primer 4 *Definisanje operatora*
; definisanje operatora deljenja tako da je x/0.0 == 0.0 za svako x
(define-fun mydiv ((x Real) (y Real)) Real
(if (not (= y 0.0))
(/ x y)
0.0))
(declare-const a Real)
(declare-const b Real)
(assert (>= (mydiv a b) 1.0))

(assert (= b 0.0))
(check-sat)

Formula predstavlja formulu nelinearne aritmetike ako je oblika (* t s), pri čemu t i s nisu linearnog oblika. Nelinearna celobrojna aritmetika je neodlučiva, tj. ne postoji procedura koja za proizvoljan ulaz vraća odgovor sat ili unsat. U najvećem broju slučajeva, Z3 vraća kao rezultat unknown. Postoji delimična podrška za nelinearnu aritmetiku zasnovana na Grebnerovim bazama. Sledi primer korišćenja Z3 rešavača za rešavanje nelinearnog problema.

Primer 5 *Upotreba nelinearne aritmetike*
(declare-const a Int)
(assert (> (a a) 3))*
(check-sat)
(get-model)
(echo „Z3 ne pronalazi uvek interpretaciju za nelinearne probleme")
(declare-const b Real)
(declare-const c Real)
(assert (= (+ (b b b) (* b c)) 3.0))*
(check-sat)
(echo „Kada postoje samo nelinearna ograničenja nad realnim konstantama, Z3 koristi posebne metode odlučivanja")
(declare-const b Real)
(declare-const c Real)
(assert (= (+ (b b b) (* b c)) 3.0))*
(check-sat)
(get-model)

Sledi primer korišćenja C++ API-ja sa nelinearom aritmetikom **??**.

```
/*
        Primer nelinearne aritmetike
*/
void nonlinear_example1() {
```

```
    config cfg;

    context c(cfg);

    expr x = c.real_const("x");
    expr y = c.real_const("y");
    expr z = c.real_const("z");

    solver s(c);

    s.add(x*x + y*y == 1);                          // x^2 + y^2 == 1
    s.add(x*x*x + z*z*z < c.real_val("1/2"));       // x^3 + z^3 < 1/2
    s.add(z != 0);
    std::cout << s.check() << "\n";
    model m = s.get_model();
}
```

Bitvektori

Postoji podrška za bit-vektore. Za razliku od programskih jezika, kao što su C, C++ i Java gde ne postoji razlika između označenih i neoznačenih bit-vektora, rešavač Z3 ih tretira na različite načine. Teorija bit-vektora ima na raspolaganju različite verzije aritmetičkih operacija za označene i neoznačene brojeve.

Z3 podržava vektore proizvoljne dužine. (_ BitVec n) je sorta bit-vektora čija je dužina n. Bit-vektor literali se mogu definisati koristeći binarnu, decimalnu ili heksadecimalnu notaciju. U binarnom i heksadecimalnom slučaju, veličina bit-vektora je određena brojem karaktera. Na primer, literal #b010 u binarnom formatu je bit-vektor dužine 3, a literal #x0a0 u heksadecimalnom formatu je bit-vektor veličine 12. Veličina bit-vektora mora biti specifikovana u decimalnom formatu. Na primer, reprezentacija (_ bv10 32) je bit-vektor dužine 32 sa vrednošću 10. Podrazumevano, Z3 predstavlja bitvektore u heksadecimalnom formatu ukoliko je dužina bitvektora umnožak broja 4 a u suprotnom u binarnom formatu. Komanda (set-option :pp.bv-literals false) se može koristiti za predstavljanje literala bitvektora u decimalnom formatu. U nastavku se navode različiti načini predstavljanja bit-vektora.

Primer 6 *Različiti načini definisanja bitvektora*

```
(display #b0100)
(display (_ bv20 8))
(display (_ bv20 7))
(display #x0a)
(set-option :pp.bv-literals false)
(display #b0100)
(display (_ bv20 8))
(display (_ bv20 7))
(display #x0a)
```

Za rad sa bit-vektorima, definisane su operacije sabiranja, oduzimanja, negacije, množenja, izračunavanja modula pri deljenju, šiftovanje u levo kao i označeno i neoznačeno šifrovanje u desno. Podržane su sledeće logičke operacije nad bitovima: disjunkcija, konjunkcija, unarna negacija, negacija konjunkcije i negacija disjunkcije. Definisane su različite relacije nad bit-vektorima kao što su <=, <, >=, > pri čemu su podržane i označene i neoznačene varijante pa se poređenje izvršava na različite načine. Primer koji sledi ilustruje pronalaženje vrednosti promenljivih korišćenjem bitvektora.

Primer 7 *Operacije nad bitvektorima*

```
(simplify (bvadd #x07 #x03)) ; dodavanje
(simplify (bvsub #x07 #x03)) ; oduzimanje
(simplify (bvneg #x07)) ; unarni minus
(simplify (bvmul #x07 #x03)) ; množenje
(simplify (bvsmod #x07 #x03)) ; označeno računanje modula
(simplify (bvshl #x07 #x03)) ; šiftovanje u levo
(simplify (bvlshr #xf0 #x03)) ; neoznačeno (logičko) šiftovanje u desno
(simplify (bvashr #xf0 #x03)) ; označeno (aritmetičko) šiftovanje u desno

(simplify (bvor #x6 #x3)) ; bitovsko ili
(simplify (bvand #x6 #x3)) ; bitovsko i
(simplify (bvnot #x6)) ; bitovska negacija
```

(simplify (bvnand #x6 #x3)) ; bitovska negacija konjunkcije
(simplify (bvnor #x6 #x3)) ; bitovska negacija disjunkcije
(simplify (bvxnor #x6 #x3)) ; bitovska negacija ekskluzivne disjunkcije

Primer 8 *Verzija De Morganovog zakona sa bit-vektorima:*
(declare-const x (_ BitVec 64))
(declare-const y (_ BitVec 64))
(assert (not (= (bvand (bvnot x) (bvnot y)) (bvnot (bvor x y)))))
(check-sat)

Ilustrujmo svojstvo aritmetike bit-vektora. Postoji brz način da se proveri da li su brojevi fiksne dužine stepeni dvojke. Ispostavlja se da je bit-vektor x stepen dvojeke ako i samo ako je vrednost x & (x - 1) jednaka 0.

Primer 9 *Provera da li je broj stepen dvojke*
(define-fun is-power-of-two ((x (_ BitVec 4))) Bool
(= #x0 (bvand x (bvsub x #x1))))
(declare-const a (_ BitVec 4))
(assert
(not (= (is-power-of-two a)
(or (= a #x0)
(= a #x1)
(= a #x2)
(= a #x4)
(= a #x8)))))
(check-sat)

Primer 10 *Relacije nad bitvektorima:*
(simplify (bvule #x0a #xf0)) ; neoznačeno manje ili jednako
(simplify (bvult #x0a #xf0)) ; neoznačeno manje
(simplify (bvuge #x0a #xf0)) ; neoznačeno veće ili jednako
(simplify (bvugt #x0a #xf0)) ; neoznačeno veće
(simplify (bvsle #x0a #xf0)) ; označeno manje ili jednako
(simplify (bvslt #x0a #xf0)) ; označeno manje
(simplify (bvsge #x0a #xf0)) ; označeno veće ili jednako
(simplify (bvsgt #x0a #xf0)) ; označeno veće

Označeno poređenje, kao što je bvsle, uzima u obzir znak bitvektora za poređenje, dok neoznačeno poređenje tretira bit-vektor kao prirodan broj.

Primer 11 *Označeno i neoznačeno poređenje bitvektora*
(declare-const a (_ BitVec 4))
(declare-const b (_ BitVec 4))
(assert (not (= (bvule a b) (bvsle a b))))
(check-sat)
(get-model)

Svojstvo da Z3 različito tretira označene i neoznačene bit-vektore ilustrovano je primerom **??**.

```
/*
 Primer sa bitvektorom pokazuje razliku u koriscenju oznacenog i neoznace
*/
void bitvector_example1() {

    context c;
    expr x = c.bv_const("x", 32);

    // koriscenje oznacenog <=
    prove((x - 10 <= 0) == (x <= 10));

    // koriscenje neoznacenog <=
    prove(ule(x - 10, 0) == ule(x, 10));
}
```

Teorija nizova

Teorija nizova

Osnovnu teoriju nizova karakterisu select i store aksiome. Komandom (select a i) vraća se vrednost na poziciji i u nizu a, a izraz (store a i v) formira novi niz, identičan nizu a pri čemu se na poziciji i nalazi vrednost v. Z3 sadrži procedure odlučivanja za osnovnu teoriju nizova. Dva niza su jednaka ukoliko su vrednosti elemenata sa

odgovarajućim indeksima jednake.

Primer 12 *Neka je a1 niz celobrojnih vrednosti. Tada je ograničenje (and (= (select a1 x) x) (= (store a1 x y) a1)) zadovoljivo kada se indeks x mapira sa vrednošću x i kada je x = y.*
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(declare-const a1 (Array Int Int))
(declare-const a2 (Array Int Int))
(declare-const a3 (Array Int Int))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(check-sat)
(get-model)
Sa druge strane, dodavanjem ograničenja (not (= x y)) izraz postaje nezadovoljiv.
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(declare-const a1 (Array Int Int))
(declare-const a2 (Array Int Int))
(declare-const a3 (Array Int Int))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(assert (not (= x y)))
(check-sat)

Konstantni nizovi

Nizovi koji sadrže konstantne vrednosti mogu se specifikovati koristeći const konstrukciju. Upotrebom const konstrukcije Z3 ne može da odluči kog tipa su elementi niza pa se on mora eksplicitno navesti.

Primer 13 *Definisanje konstantnog niza koji sadrži samo jedinice*
(declare-const all1 (Array Int Int))

(declare-const a Int)
(declare-const i Int)
(assert (= all1 ((as const (Array Int Int)) 1)))
(assert (= a (select all1 i)))
(check-sat)
(get-model)
(assert (not (= a 1)))
(check-sat)

Interpretacija nizova je slična interpretaciji funkcija. Z3 koristi konstrukciju (_ as-array f) za određivanje interpretacije niza. Ako je niz a jednak rezultatu konstrukcije (_ as-array f), tada za svaki indeks i, vrednost (select a i) odgovara vrednosti (f i).

Primena map funkcije na nizove Z3 nudi parametrizovanu funkciju map na nizove. Omogućava primenu proizvoljnih funkcija na sve elemente niza.

Primer 14 *Primena map funkcije na nizove*
(define-sort Set (T) (Array T Bool))
(declare-const a (Set Int))
(declare-const b (Set Int))
(declare-const c (Set Int))
(declare-const x Int)
(push)
(assert (not (= ((_ map and) a b) ((_ map not) ((_ map or) ((_ map not) b) ((_ map not) a))))))
(check-sat)
(pop)
(push)
(assert (and (select ((_ map and) a b) x) (not (select a x))))
(check-sat)
(pop)
(push)
(assert (and (select ((_ map or) a b) x) (not (select a x))))
(check-sat)
(get-model)

(assert (and (not (select b x))))
(check-sat)

Tipovi podataka

Algebarski tipovi podataka omogućavaju specifikaciju uobičajnih struktura podataka. Slogovi i taplovi su specijalne vrste algebarskih tipova podataka kao i skalari (enumeracijski tipovi). Primena algebarskih tipova podataka može se generalizovati. Mogu se koristiti za specifikovanje konačnih lisi, stabala i rekurzivnih struktura.
Slogovi
Slog se specifikuje kao tip podataka sa jednim konstruktorom i proizvoljnim brojem elemenata sloga. Sistem ne dozvoljava proširivanje slogova. Naredni primer ilustruje da su dva sloga jednaka samo ako su im svi argumenti jednaki. Uvodimo parametarski tip Pair, sa konstruktorom mk-pair i dva argumenta kojima se može pristupiti koristeći selectorske funkcije first i second.

Primer 15 *Definisanje sloga kao tipa podataka*
(declare-datatypes (T1 T2) ((Pair (mk-pair (first T1) (second T2)))))
(declare-const p1 (Pair Int Int))
(declare-const p2 (Pair Int Int))
(assert (= p1 p2))
(assert (> (second p1) 20))
(check-sat)
(get-model)
(assert (not (= (first p1) (first p2))))
(check-sat)

Skalari (tipovi enumeracije)

Sorta skalara je sorta konačnog domena. Elementi konačnog domena se tretiraju kao različite konstante. Na primer, sorta S je skalarni tip sa tri vrednosti A, B i C. Moguće je da tri konstante sorte S budu različite. Ovo svojstvo ne može važiti u slučaju četiri konstante.

Primer 16 *Korišćenje skalarnog tipa podataka*
(declare-datatypes () ((S A B C)))
(declare-const x S)

(declare-const y S)
(declare-const z S)
(declare-const u S)
(assert (distinct x y z))
(check-sat)
(assert (distinct x y z u))
(check-sat)

Rekurzivni tipovi podataka
Deklaracija rekurzivnog tipa podataka uključuje sebe direktno kao komponentu. Standardni primer rekurzivnog tipa podataka je lista. Parametrizovana lista može se definisati na sledeći način:
(declare-datatypes (T) ((Lst nil (cons (hd T) (tl Lst)))))
(declare-const l1 (Lst Int))
(declare-const l2 (Lst Bool))

Postoji podrška za rekurzivni tip podataka korišćenjem ključne reči List. Prazna lista se definiše korišćenjem reči nil a konstruktor insert se koristi za formiranje novih lista. Selektori head i tail se definišu na uobičajan način.

Primer 17 *Korišćenje ugrađene podrške za liste*
(declare-const l1 (List Int))
(declare-const l2 (List Int))
(declare-const l3 (List Int))
(declare-const x Int)
(assert (not (= l1 nil)))
(assert (not (= l2 nil)))
(assert (= (head l1) (head l2)))
(assert (not (= l1 l2)))
(assert (= l3 (insert x l2)))
(assert (> x 100))
(check-sat)
(get-model)
(assert (= (tail l1) (tail l2)))
(check-sat)

U prethodnom primeru, uvodi se ograničenje da su liste l1 i l2 različite od nil. Ova ograničenja se uvode jer iterpretacija selektora head i tail nije specifikovana u slučaju nedefinisanih lista. Tada pomenuti selektori neće moći da razlikuju nil od komande (insert (head nil) (tail nil)).

Glava 3

Zaključak

Bibliography

[1] Yuri Gurevich and Saharon Shelah. “Expected computation time for Hamiltonian path problem”. English. In: *SIAM Journal on Computing* 16 (1987), pp. 486–502.

[2] Petar Petrović and Mika Mikić. „Naučni rad”. serbian. In: *Konferencija iz matematike i računarstva*. Ed. by Miloje Milojević. 2015.

Biografija autora

Vuk Stefanović Karadžić (*Tršić, 26. oktobar/6. novembar 1787. — Beč, 7. februar 1864.*) bio je srpski filolog, reformator srpskog jezika, sakupljač narodnih umotvorina i pisac prvog rečnika srpskog jezika. Vuk je najznačajnija ličnost srpske književnosti prve polovine XIX veka. Stekao je i nekoliko počasnih mastera. Učestvovao je u Prvom srpskom ustanku kao pisar i činovnik u Negotinskoj krajini, a nakon sloma ustanka preselio se u Beč, 1813. godine. Tu je upoznao Jerneja Kopitara, cenzora slovenskih knjiga, na čiji je podsticaj krenuo u prikupljanje srpskih narodnih pesama, reformu ćirilice i borbu za uvođenje narodnog jezika u srpsku književnost. Vukovim reformama u srpski jezik je uveden fonetski pravopis, a srpski jezik je potisnuo slavenosrpski jezik koji je u to vreme bio jezik obrazovanih ljudi. Tako se kao najvažnije godine Vukove reforme ističu 1818., 1836., 1839., 1847. i 1852.