

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

1	Uvod	1
2	Razrada	2
2.1	Rešavač Z3	2
3	Zaključak	18
	Bibliography	19

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 teorema kompanije Microsoft koji se može koristiti za proveru zadovoljivosti logičkih formula u velikom broju teorija. Neke od njih su teorija linearne aritmetike, teorija bitvektora, teorija neinterpretiranih 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 bez parametara. 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 neinterpretirani. Ovo je kontrast u odnosu na funkcije odgovarajućih teorija. Funkcija $+$ ima standardnu iterpretaciju u teoriji aritmetike. Neinterpretirane 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 .

```
/*
  Kod demonstrira upotrebu neinterpretiranih tipova
  i funkcija
  dokazivanjem jednakosti:
   $x = y \Rightarrow g(x) = g(y)$ 
*/
void dokazi_jednakost() {

    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 << "dokazano";  
else  
    std::cout << "ne moze se dokazati";  
}
```

Komandom `declare-const` deklarirše 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` deklarirše se funkcija.

Primer 1 *Naredni kod demonstrira upotrebu konstanti i funkcija. U primeru se deklarirše konstanta a celobrojnog tipa i funkcija f sa parametrima tipa Int i $Bool$ i povratnom vrednošću celobrojnog tipa. Zatim se dodaju odgovarajuća ograničenja za konstantu a i funkciju f korišćenjem operatora poređenja. Rešavač Z3 pronalazi da je ovo tvrđenje zadovoljivo i daje prikazani model.*

<i>(declare-const a Int)</i>	<i>sat (model</i>
<i>(declare-fun f (Int Bool) Int)</i>	<i>(define-fun a () Int 11)</i>
<i>(assert (> a 10))</i>	<i>(define-fun f ((x!1 Int) (x!2</i>
<i>(assert (< (f a true) 100))</i>	<i>Bool)) Int</i>
<i>(check-sat)</i>	<i>(ite (and (= x!1 11) (= x!2 true))</i>
<i>(get-model)</i>	<i>0 0))</i>
	<i>)</i>

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 dualnosti ispitivanjem validnosti formule: $\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b)$ tako što se kao ograničenje dodaje negacija polazne formule. Z3 pronalazi da je ovo tvđenje nezadovoljivo, pa je formula tačna u svim interpretacijama.*

```
(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)
(get-model)
```

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

```
/**
```

Dokazivanje De Morganovog zakona dualnosti:

```
{not(x and y) <-> (not x) or ( not y) }
```

```
*/
```

```
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 \sim (unarna negacija). Konstante se mogu porediti korišćenjem operatora $=$, $<$, $<=$, $>$, $>=$. U celobrojnoj linearnoj aritmetici, interpretirani funkcijski simboli su $+$, $-$, $*$, $\% \bmod$ i \div . U slučaju funkcija \bmod 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 *Naredni kod ilustruje pronalaženje interpretacija celobrojnih i realnih konstanti. Interpretacija se svodi na pridruživanje brojeva svakoj konstanti u slučaju zadovoljivosti formule. Ograničenja sadrže pomenute aritmetičke operatore. Realne konstante treba da budu zapisane sa decimalnom tačkom. Zbog pomenutog svojstva da Z3 ne vrši automatsku konverziju između celobrojnih i realnih konstanti, koristimo funkciju `to-real` za konvertovanje u realne vrednosti. Procedura odlučivanja vraća zadovoljivost tvrdjenja i dobijeni model prikazujemo u nastavku.*

```
(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Real)
(declare-const e Real)
(assert (> e (+ (to_real (+ a b))
2.0)))
(assert (= d (+ (to_real c) 0.5)))
(assert (> a b))
(check-sat)
(get-model)

sat
(model
(define-fun b () Int 0)
(define-fun a () Int 1)
(define-fun e () Real 4.0)
(define-fun c () Int 0)
(define-fun d () Real (/ 1.0 2.0))
)
```

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 *Kod definiše operator deljenja tako da rezultat bude specifikovan i kada je delilac 0. Zatim se uvode dve konstante realnog tipa i primenjuje se definisani operator. Z3 rešavač pronalazi nezadovoljivost tvrdjenja, s obzirom da operator mydiv vraća 0 pa relacija poređenja ne može biti tačna.*

```
; definišemo da je x/0.0 == 0.0          unsat
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. Postoje nelinearni problemi za koje Z3 rešavač vraća odgovarajući model. Postoji delimična podrška za nelinearnu aritmetiku zasnovana na Grebnerovim bazama.

Primer 5 *Rešavanje različitih nelinearnih problema sa celobrojnim i realnim konstantama. Z3 rešavač ne pronalazi uvek model za nelinearne probleme, ovde se za drugo tvrđenje vraća nezadovoljivost. Kada su prisutna samo nelinearna ograničenja nad realnim konstantama, Z3 koristi posebne metode odlučivanja. Ove metode korišćene su za dokazivanje trećeg tvrđenja.*

<code>(declare-const a Int)</code>	<code>sat</code>
<code>(assert (> (* a a) 3))</code>	<code>(model</code>
<code>(check-sat)</code>	<code>(define-fun a () Int (- 8)))</code>
<code>(get-model)</code>	<code>unsat</code>
<code>(declare-const b Real)</code>	<code>sat (</code>
<code>(declare-const c Real)</code>	<code>model</code>
<code>(assert (= (+ (* b b b) (* b c))</code>	<code>(define-fun b () Real (/ 1.0 8.0))</code>
<code>3.0))</code>	<code>(define-fun c () Real (/ 1535.0</code>
<code>(check-sat)</code>	<code>64.0)))</code>
<code>(declare-const b Real)</code>	
<code>(declare-const c Real)</code>	
<code>(assert (= (+ (* b b b) (* b c))</code>	
<code>3.0))</code>	
<code>(check-sat)</code>	
<code>(get-model)</code>	

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

Z3 podržava vektore proizvoljne dužine. (`_ BitVec n`) je sorta bitvektora čija je dužina `n`. Bitvektor literali se mogu definisati koristeći binarnu, decimalnu ili heksadecimalnu notaciju. U binarnom i heksadecimalnom slučaju, veličina bitvektora je određena brojem karaktera. Na primer, literal `#b010` u binarnom formatu je bitvektor dužine 3, a literal `#x0a0` u heksadecimalnom formatu je bitvektor veličine 12. Veličina bitvektora mora biti specificovana u decimalnom formatu. Na primer, reprezentacija (`_ bv10 32`) je bitvektor 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.

Primer 6 *Navodimo različite načine predstavljanja bitvektora. Ukoliko zapis počinje sa #b, bitvektor se zapisuje u binarnom formatu. Ukoliko zapis počinje sa #x, bitvektor se zapisuje u heksadecimalnom formatu. U oba slučaja, nakon specifikacije formata, zapisuje se dužina vektora. Drugi način zapisa počinje skraćenicom bv, navođenjem vrednosti i na kraju dužine.*

<code>(display #b0100)</code>	<code>#x4</code>
<code>(display (_ bv20 8))</code>	<code>#x14</code>
<code>(display (_ bv20 7))</code>	<code>#b0010100</code>
<code>(display #x0a)</code>	<code>#x0a</code>
<code>(set-option :pp.bv-literals false)</code>	<code>(_ bv4 4)</code>
<code>(display #b0100)</code>	<code>(_ bv20 8)</code>
<code>(display (_ bv20 8))</code>	<code>(_ bv20 7)</code>
<code>(display (_ bv20 7))</code>	<code>(_ bv10 8)</code>
<code>(display #x0a)</code>	

Za rad sa bitvektorima, 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 bitvektorima kao što su \leq , $<$, \geq , $>$ 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 *Ilustracija podržanih aritmetičkih operacija nad bitvektorima uključujući sabiranje, oduzimanje, unarnu negaciju, množenje, računanje modula, šiftovanje ulevo, neoznačeno (logičko) šiftovanje udesno i označeno (aritmetičko) šiftovanje udesno.*

<code>(simplify (bvadd #x07 #x03))</code>	<code>#x0a</code>
<code>(simplify (bvsb #x07 #x03))</code>	<code>#x04</code>
<code>(simplify (bvneg #x07))</code>	<code>#xf9</code>
<code>(simplify (bvmul #x07 #x03))</code>	<code>#x15</code>
<code>(simplify (busmod #x07 #x03))</code>	<code>#x01</code>
<code>(simplify (bushl #x07 #x03))</code>	<code>#x38</code>
<code>(simplify (bulshr #xf0 #x03))</code>	<code>#x1e</code>
<code>(simplify (bvashr #xf0 #x03))</code>	<code>#xfe</code>
<code>(simplify (bvor #x6 #x3))</code>	<code>#x7</code>
<code>(simplify (bvand #x6 #x3))</code>	<code>#x2</code>
<code>(simplify (bvnot #x6))</code>	<code>#x9</code>
<code>(simplify (bvnan #x6 #x3))</code>	<code>#xd</code>
<code>(simplify (bvnor #x6 #x3))</code>	<code>#x8</code>
<code>(simplify (bvxnor #x6 #x3))</code>	<code>#xa</code>

Primer 8 Naredni kod dokazuje validnost De Morganovog zakona korišćenjem bitvektora. Deklarišu se dve konstante predstavljene bitvektorima dužine 64 a zatim se dodaje negacija formule i ispituje se njena zadovoljivost. Rešavač Z3 vraća *unsat*, negacija formule je nezadovoljiva. Oдавде zaključujemo da je polazna formula valjana.

```

(declare-const x (_ BitVec 64))                                unsat
(declare-const y (_ BitVec 64))
(assert
(not (= (bvand (bvnot x) (bvnot y)) (bvnot
(bvor x y))))))
(check-sat)
(get-model)

```

Ilustrujmo svojstvo aritmetike bitvektora. Postoji brz način da se proverí da li su brojevi fiksne dužine stepeni dvojke. Ispostavlja se da je bitvektor x stepen dvojke ako i samo ako je vrednost izraza $x \& (x - 1)$ jednaka 0.

Primer 9 Provera da li je broj stepen dvojke vrši se definisanjem funkcije korišćenjem prethodno pomenute jednakosti. Dodaje se negacija ove jednakosti kao tvrdjenja i vrši se proveravanje za bitvektore vrednosti 0, 1, 2, 4 i 8. U svim slučajevima brojevi su stepeni dvojke pa Z3 rešavač vraća nezadovoljivost.


```
(define-fun is-power-of-two ((x (_ BitVec 4))) Bool
  (= #x0 (bvand x (bvsb 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)
(get-model)
```

Primer 10 *Ilustracija podržanih relacija nad bitvektorima. Podržane relacije uključuju neoznačene i označene verzije za operatore $<$, $<=$, $>$ i $>=$. U nastavku slede pomenuti operatori primenom na dva operanda i njihove vrednosti kao izlaze.*

(simplify (bvule #x0a #xf0))	true
(simplify (bvult #x0a #xf0))	true
(simplify (bvuge #x0a #xf0))	false
(simplify (bvugt #x0a #xf0))	false
(simplify (bvule #x0a #xf0))	false
(simplify (bvult #x0a #xf0))	false
(simplify (bvuge #x0a #xf0))	true
(simplify (bvugt #x0a #xf0))	true

Primer 11 *Ilustracija označenog i neoznačenog poredenja bitvektora. Označeno poredenje, kao što je bvule, uzima u obzir znak bitvektora za poredenje, dok neoznačeno poredenje tretira bitvektor kao prirodan broj. Z3 rešavač pronalazi da je tvrdjenje zadovoljivo i daje prikazani model.*

<code>(declare-const a (_ BitVec 4))</code>	<code>sat</code>
<code>(declare-const b (_ BitVec 4))</code>	<code>(model</code>
<code>(assert (not (= (bvule a b) (bvule a b))))</code>	<code>(define-fun b () (_ BitVec 4) #xe)</code>
<code>(check-sat)</code>	<code>(define-fun a () (_ BitVec 4) #x0))</code>
<code>(get-model)</code>	

Svojstvo da Z3 različito tretira označene i neoznačene bitvektore ilustrovano je primenom.

```

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

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

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

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

```

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 Definišemo tri konstante x , y i z celobrojnog tipa. Neka je $a1$ niz celobrojnih vrednosti. Tada je ograničenje $(\text{and } (= (\text{select } a1\ x)\ x) (= (\text{store } a1\ x\ y)\ a1))$ zadovoljivo kada je element niza a na poziciji x jednak definisanoj konstanti x

i u slučaju kada su konstante x i y jednake. Rešavač Z3 vraća zadovoljivost zadatog tvrđenja i odgovarajući model.

<i>(declare-const x Int)</i>	<i>sat</i>
<i>(declare-const y Int)</i>	<i>(model</i>
<i>(declare-const z Int)</i>	<i>(define-fun y () Int 1)</i>
<i>(declare-const a1 (Array Int Int))</i>	<i>(define-fun a1 () (Array Int Int) (_ as-</i>
<i>(assert (= (select a1 x) x))</i>	<i>array k!0))</i>
<i>(assert (= (store a1 x y) a1))</i>	<i>(define-fun x () Int 1)</i>
<i>(check-sat)</i>	<i>(define-fun k!0 ((x!1 Int)) Int</i>
<i>(get-model)</i>	<i>(ite (= x!1 1) 1</i>
	<i>0)))</i>

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. 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)`.

Primer 13 *Defnišemo konstantni niz `all1` celobrojnog tipa i dve celobrojne konstante `a` i `i`. Uvodimo ograničenje da niz `all1` sadrži samo jedinice. Z3 pronalazi da je ovo tvrdjenje zadovoljivo, i daje prikazani model.*

<i>(declare-const all1 (Array Int Int))</i>	<i>sat</i>
<i>(declare-const a Int)</i>	<i>(model</i>
<i>(declare-const i Int)</i>	<i>(define-fun all1 () (Array Int Int)</i>
<i>(assert (= all1 ((as const (Array Int Int))</i>	<i>(_ as-array k!0))</i>
<i>1)))</i>	<i>(define-fun i () Int 0)</i>
<i>(assert (= a (select all1 i)))</i>	<i>(define-fun a () Int 1)</i>
<i>(check-sat)</i>	<i>(define-fun k!0 ((x!0 Int)) Int</i>
<i>(get-model)</i>	<i>(ite (= x!0 0) 1 1)))</i>

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

Primer 14 Definišemo dva konstantna niza a i b tipa *Boolean* i dokazujemo da važi svojstvo $\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b)$ primenom na sve elemente nizova a i b korišćenjem funkcije *map*. Kao ograničenje dodajemo negaciju prethodno navedene formule. Rešavaš *Z3* vraća nezadovoljivost negirane formule, odakle zaključujemo da je polazna formula validna.

```
(define-sort Set (T) (Array T Bool))                                unsat
(declare-const a (Set Int))
(declare-const b (Set Int))
(assert (not (= ((_ map and) a b)
  ((_ map not)
    ((_ map or) ((_ map not) b) ((_ map
not) a)))))))
(check-sat)
```

Tipovi podataka

Algebarski tipovi podataka omogućavaju specifikaciju uobičajnih struktura podataka. Slogovi i torke 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. Važi svojstvo da su dva sloga jednaka samo ako su im svi argumenti jednaki.

Primer 15 Pokazujemo svojstvo da su dva sloga jednaka ako i samo ako su im svi argumenti jednaki. Uvodimo parametarski tip *Pair*, sa konstruktorom *mk-pair* i dva argumenta kojima se može pristupiti koristeći selektorske funkcije *first* i *second*. Definišemo dva sloga $p1$ i $p2$, čija su oba podatka celobrojnog tipa. Dodajemo ograničenja da su slogovi $p1$ i $p2$ jednaki kao i ograničenje koje se odnosi na drugi element

sloga. Rešavač Z3 u prvom slučaju vraća zadovoljivost formule i odgovarajući model. Dodavanjem ograničenja da prvi elementi slogova nisu jednaki korišćenjem selektorske funkcije *first*, tvđenje postaje nezadovoljivo.

<i>(declare-datatypes (T1 T2) ((Pair (mk-</i>	<i>sat</i>
<i>pair (first T1) (second T2))))</i>	<i>(model</i>
<i>(declare-const p1 (Pair Int Int))</i>	<i>(define-fun p1 () (Pair Int Int)</i>
<i>(declare-const p2 (Pair Int Int))</i>	<i>(mk-pair 0 21))</i>
<i>(assert (= p1 p2))</i>	<i>(define-fun p2 () (Pair Int Int)</i>
<i>(assert (> (second p1) 20))</i>	<i>(mk-pair 0 21)))</i>
<i>(check-sat)</i>	<i>unsat</i>
<i>(get-model)</i>	
<i>(assert (not (= (first p1) (first p2))))</i>	
<i>(check-sat)</i>	

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 *Prilikom deklaracije skalarnog tipa podataka, navodi se broj različitih elemenata domena, u ovom primeru tri i pokazuje se nezadovoljivost tvđenja sa četiri različita elementa domena.*

<i>(declare-datatypes () ((S A B C)))</i>	<i>sat</i>
<i>(declare-const x S)</i>	<i>(model</i>
<i>(declare-const y S)</i>	<i>(define-fun z () S A)</i>
<i>(declare-const z S)</i>	<i>(define-fun y () S B)</i>
<i>(declare-const u S)</i>	<i>(define-fun x () S C))</i>
<i>(assert (distinct x y z))</i>	<i>unsat</i>
<i>(check-sat)</i>	
<i>(assert (distinct x y z u))</i>	
<i>(check-sat)</i>	

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. Deklarišemo tri liste l1, l2 i l3 sa celobrojnim vrednostima, kao i celobrojnu konstantu x. Dodaju se ograničenja za prve i poslednje elemente liste korišćenjem selektora.*

<code>(declare-const l1 (List Int))</code>	<code>sat</code>
<code>(declare-const l2 (List Int))</code>	<code>(model</code>
<code>(declare-const x Int)</code>	<code>(define-fun l3 () (List Int)</code>
<code>(assert (not (= l1 nil)))</code>	<code>(insert 101 (insert 0 (insert 1 nil))))</code>
<code>(assert (not (= l2 nil)))</code>	<code>(define-fun x () Int 101)</code>
<code>(assert (= (head l1) (head l2)))</code>	<code>(define-fun l1 () (List Int) (insert 0 nil))</code>
<code>(assert (not (= l1 l2)))</code>	<code>(define-fun l2 () (List Int) (insert 0 (in-</code>
<code>(assert (= l3 (insert x l2)))</code>	<code>sert 1 nil))))</code>
<code>(assert (> x 100))</code>	<code>unsat</code>
<code>(check-sat)</code>	
<code>(get-model)</code>	
<code>(assert (= (tail l1) (tail l2)))</code>	
<code>(check-sat)</code>	

U prethodnom primeru, uvodi se ograničenje da su liste l1 i l2 različite od nil. Ova ograničenja se uvode jer interpretacija 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.