PRIMENA SIMBOLIČKOG IZVRŠAVANJA PROGRAMA U TESTIRANJU PROGRAMA
APPLYING OF SYMBOLIC PROGRAM EXECUTION IN PROGRAM TESTING

Branko Markoski, Dragica Radosav, *Technical Faculty "M. Pupin" Zrenjanin, University of Novi Sad, Serbia*
Jovan Šetrajčić, *Department of Physics, Faculty of Sciences, University of Novi Sad, Serbia*
Branko Petrevski, *BS of Sremska Mitrovica, Serbia*
Miroslava Petrevska, *Faculty of Economics Subotica, University of Novi Sad, Serbia*
Milošević Zoran, *5University of Novi Sad, Faculty of Sports and Physical Education, Serbia*

Sadržaj - *Poslednjih godina porasla je aktivnost na polju verifikacije programa. Cilj ovih napora je da se konsrtruisu kompjuterski sistemi za određivanje je li dati kompjuterski program ispravan, u smislu zadovoljavanja datih specifikacija. Testiranje programa je u principu komplikovani proces koji se mora izvršiti što sistematičnije moguće , kako bi pružio adekvatnu pouzdanost i uverenje o kvalitetu. Testiranje programa se često povezuje sa otkrivanjem " bagova ". Specifikacija je ključni stvar u testiranju programa. Način na koji program reaguje na različite ulazne podatke propisan je specifikacijom. Ako je u opštem slučaju ulaz u program označen sa x, a njegov izlaz sa y tada specifikacija može da se shvati kao relacija koja povezuje skup ulaza sa skupom izlaza, pri čemu se pojmovi "ulaz" i "izlaz" shvataju u najširem smislu reči. Aktivnost testiranja pokazuje da li je softver usklađen sa specifikacijom. Najstarija i najpoznatija metoda koja se primenjuje za konstruktivno testiranje manjih programa je simboličko izvršavanje programa.Ova metoda je možda nedovoljno poznata po svom nazivu, ali je zato poznata po svim svojim prednostima. Jedan od načina njene primene je putem računara što sa druge strane nije osnovni uslov. Samo donekle podseća na metodu simulacije rada računara, ali se od nje bitno razlikuje po cilju. Osnovni cilj ove metode je konstruktivno testiranje.*

Ključne reči - *testiranje, specifikacija, validacija, test*

Abstract - *In recent years we have increasing activity in the field of program verification. The goal of these efforts is to construct computer systems to determine whether the computer program is correct, in terms of meeting the given specifications. Program testing is in principle a complicated process that must be done as systematically as possible, to provide adequate reliability and quality certificate. Program testings are often associated with detecting "bugs". Specification is the key thing in the program testing. The way the program reacts to different input data is prescribed by the specification. If in the general, entrance to the program is represented by x, and its output with y then specifications can be understood as a relation that connects a set of inputs with a set of outputs, while the terms "input" and "output" can be understand in the broadest sense of the word. Testing activity show does the software complies with the specification. The oldest and best known method that is used for constructive testing smaller program is symbolic execution of programs. This method may not be known by his name, but it is known for all its advantages. One way of its use is by computer, which is not primary condition. It may look like computer simulation method, but it significantly differs from it by its goal. The main goal of this method is constructive testing.*

Key words - *testing, specification, validation, test*

1. UVOD

U životnom ciklusu softvera testiranje programa predstavlja vrlo značajnu, a ako ne i najvažniju aktivnost jer se mora potvrditi kvalitet zahteva specifikacije, dizajna i primene. Po Petersu (1) , velike i ozbiljne firme troše oko 40 % za testiranje programa. Da bi se izvelo testiranje velikih i složenih programa ono se mora izvesti što sistematičnije, kako bi se dobila pouzdanost. Ako se radi o testiranju velikih sistema, odnosno njihovog operativnog sistema najčešće se prilazi ad

hok testiranju koji ne može najčešće da potvrdi ni kvalitet ni ispravnost prema specifikaciji ni prema konstrukciji ni prema primeni. U osnovi svih definicija testiranja programa je težnja da se odgovori na pitanje: da li se program ponaša onako kako je zahtevano? U radovima Džemjsa Vitakera (2) se može videti zašto je testiranje programa teško Validacija i verifikacija su izrazi koji se najčešće povezuju sa testiranjem programa.Verifikacija predstavlja proveru ili testiranje objekata (ili programa) u cilju utvrđivanja koliko odgovaraju predviđenim karakteristikama. Verifikacija obuhvata analize, inspekciju, isprobavanje, a jedan od njenih oblika je i testiranje programa.Verifikacija programa se može osloniti na tehnike automatskog dokazivanja teoreme. Ove tehnike otelotvoruju principe deduktivnog zaključivanja, iste one koje koriste i programeri prilikom samog konstruisanja programa. Zašto ne bi koristili iste principe u sistemu za automatsku sintezu, koji može da konstruiše program umesto da samo dokazuje njegovu ispravnost? Svakako, konstruisanje programa zahteva više originalnosti i kreativnosti nego dokazivanje njegove ispravnosti, ali oba posla zahtevaju isti način razmišljanja. Ukoliko se detektuju greške ili problemi u računaru najčešće se koristi izraz bag. Ovaj izraz se primarno koristio u SAD, i to u vreme kada su računari izgledali "primitivno" i kada su određeni problemi bili prouzrokovani postojanjem živih bubica. Postoje više podela bagova, Jorg (3) ali napoznatija podela je na hardverske i softverske. Kada se testiranjem programa utvrdi da se on ne ponaša kao što se očekuje, pristupa se analiziranju i lociranju bagova. Detaljan postupak indentifikaciji bagova omogućuje se metodičkim pristupom testiranju programa. Potraga za bagovima olakšava testiranje programa, ali ne može adekvatno da ga zameni. Poznato je takođe da nikakvo testiranje ne može da otkrije sve bagove. U vezi sa testiranjem programa često se rade statičke analize (istražuju se osnovni programi, pri tome traže se osnovni problemi i prikupljaju podaci bez izvršavanja programa) i dinamičke analize (istražuje se ponašanje programa u izvršenju i tako se dobijaju podaci vezani za puteve izvršavanja, vremenske profile i pokrivenosti testiranja).

2. SIMBOLIČKO IZVRŠAVANJE PROGRAMA

Prihvaćeni način za dokazivanje stvari u programima je koričćenje pravila zaključivanja kao što ih je izneo Hoare (4). Ova pravila se obično formulišu tako da se mogu primeniti na poslednju programsku liniju, čime se proverava jedan ili više kratkih programa i možda neke logičke formule. Iterativnom primenom pravila zaključivanja, zadatak dokazivanja ispravnosti programa svodi se na dokazivanje programskih linija u računu verovatnoće.

Druga tehnika dokazivanja je zasnovana na pojmu simboličnog izvršenja: programske linije se obrađuju istim redom kao što će biti izvršavane, za razliku od redosleda "unatraške" kod prvog metoda. Izgleda da su obe tehnike jednako snažne i logički ekvivalentne. No, bliska analogija između simboličkog interpretativnog izvršenja izgleda da čini metod simbolikog izvršenja lakšim za razumevanje. Jedna od najstarijih i najpoznatija metoda koja se primenjuje za konstruktivno testiranje manjih programa je simboličko izvršavanje programa.Ova metoda je možda nedovoljno poznata po svom nazivu, ali je zato poznata po svim svojim prednostima. Jedan od načina njene primene je putem računara što sa druge strane nije osnovni uslov. Samo donekle podseća na metodu simulacije rada računara, ali se od nje bitno razlikuje po cilju. Osnovni cilj ove metode je konstruktivno testiranje.

Pod pojmom uspešnog testa mogu se podrazumevati dva, i to suprotna koncepta, što je i od najveće važnosti za opšti pristup testiranju. Posmatrano sa stanovišta izrade korektnog programa, test se smatra uspešnim ukoliko nema grešaka. No, međutim, ukoliko se uzme u obzir utrošeno vreme kao i uloženi novac na testiranje sasvim je opravdano diskutovati o uspešnom testu u smislu dokaza da grešaka ima. S obzirom na to da i jedan i drugi prilaz imaju smisla, u prvom slučaju govorimo o *konstruktivnom* pristupu testiranju, a u drugom o *destruktivnom* pristupu testiranju i isto tako o konstruktivno uspešnom i destruktivno uspešnom testu.

U simboličkom izvršenju, vrednosti promenljivih u programu predstavjaju se simboliškim konstantama ili izrazima. Na primer, vrednost promenljive V može se predstaviti kao "B + 3", gde je B simbolička konstanta (a ne promenljiva programa). Skup svih promenljivih i njihovih vrednosti naziva se izjava po J. Vass-u. Dok se programska putanja "izvršava", pretpostavke o stanju se čuvaju u uslovima putanje.

Princip metode se sastoji u tome da se ulaznim promenljivim umesto konkretnih dodeljuju simboličke vrednosti nakon čega se zatim simulacijom bez računara ili primenom odgovarajućih programa specijalne namene odrede simboličke formule za izlaze koje se na kraju upoređuju sa specifikacijom. Držeći se osnovnih principa metode može se postići da jedan simbolički ulaz pokrije veći, pa i veoma velik broj konkretnih. Napr. deo Pascal programa koji prema specifikaciji treba da računa $y=(a^2-b^2)*x$ izgleda ovako:

```
readln(x,a,b);          {1}
y:=x*b;                 {2}
x:=x*a;                 {3}
y:=(x-y)*(a+b);         {4}
writeln(y);
```

ovde su x, y, a i b realne promenljive u jednostrukom formatu (tip *single* koji je najkraći u poređenju sa ostalim realnim tipovima u Pascalu). Dokaz da je programski segment korektan(izvršavanjem programa za različite konkretne vrednosti ulaza), zahteva formiranje test skupa T koji sadrži sve moguće kombinacije svih mogućih vrednosti ulaznih promenjivih x, a i b. Ako je R skup svih *single* konstanti tada bi domen test skupa T iznosio:

$$dom(T)=R^3.$$

Prema standardu, tip *single* (jednostruki realni tip) ima: 1 bit predznaka, e=8 bitova eksponenta i m=23 bitova za mantisu. Obzirom da je i normalizovan, ukupan broj konstanti ovog tipa iznosi

$$\frac{2^{m+1} * 2^e - 2^{e+1}}{2} + 1 = 2^m * 2^e - 2^e + 1 = 2^e * (2^m - 1) + 1$$

pa je, prema tome, broj elemenata domena skupa T dat sa

$$\left|dom(T)\right| = \left[2^e * (2^m - 1) + 1\right]^3$$

Za navedene vrednosti m i e veličina dom(T) iznosi približno 10^{28} različitih ulaznih trojki (x,a,b)!

Kod simboličkog izvršavanja zadaju se simboli koji su predstavnici čitave klase konkretnih ulaznih vrednosti umesto konkretnih ulaza .To je u ovom slučaju konstanta tipa *single* .Umesto realnih konstanti u ovom slučaju zadaju se simbolički ulazni podaci:

x=X a=A b=B

X, A i B su bilo koje konstante tipa *single*(bilo koji elementi skupa R). Ukoliko se dokaže da nakon izvršenja segmenta promenljiva y dobija vrednost $(A^2-B^2)*X$ to važi za *ma koju trojku elemenata iz skupa R* .Na taj način je 10^{28} konkretnih ulaznih trojki zamenjeno jednom jedinom simboličkom. Naredbe označene sa 1, 2, 3 i 4 simboličkim izvršavanjem daju sledeću sekvencu jednakosti:

```
x=X    a=A    b=B          {1}
y=X*B                      {2}
x=X*A                      {3}
y=(X*A-X*B)*(A+B)          {4}
```

Do dokaza da je program korektan dolazi se prostom transformacijom poslednjeg izraza $y=X*(A-B)*(A+B)=X*(A^2-B^2)$.

Pored niza prednosti koje ova metoda ima, osnovni nedostaci su vezani za naredbe tipa *if* odnosno *case* a naročito sa naredbama repeticije (ciklusa). O čemu se zapravo radi? Pored simboličkih vrednosti promenljivih u oba slučaja moraju se uvoditi dodatne pretpostavke tako da budu predviđeni svi različiti ishodi. I to kod naredbi tipa *if-then-else* oba ishoda, a kod repeticije i nastavak i izlaz iz ciklusa. Primer za programsku sekvencu oblika:

```
read(a,b,c);
if abs(a)>abs(b) then c:=a
        else c:=b;
write(c);
```

predviđaju se dva simbolička ulaza:

$a = A \quad b = B$ uz pretpostavku $\left|A\right| > \left|B\right|$

$a = A \quad b = B$ uz pretpostavku $\left|A\right| \le \left|B\right|$

Postupak simboličkog izvršavanja postaje neupotrebljiv kada su predikati koji kontrolišu izvršavanje selekcije složeni.. Još je složenija situacija sa repeticijama kojih ima u svakom ozbiljnijem programu bar na stotine, jer se kod njih moraju predvideti različite varijante završavanja. Formule po kojima se računaju izlazne promenljive mogu biti, i jesu, vrlo složene, tako da i matematički aspekt testiranja može da bude veoma kompleksan.

Sve su ovo razlozi zbog kojih se simboličko izvršavanje programa koristi u posebnim okolnostima među kojima su:

- Testiranje kritičnih delova programa
- Zajedno sa drugim metodama
- Uz primenu specijalnih programa za simboličko izvršavanje
- Zajedno sa kombinovanjem simboličkih i konkretnih vrednosti ulaza, gde se konkretne vrednosti biraju na osnovu klasa sličnosti i graničnih vrednosti
- Uz upotrebu posebnih matematičkih tehnika za obradu pojedinih programskih upravljačkih struktura

Primena posebnih matematičkih tehnika za dokazivanje korektnosti programskih sekvenci koje sadrže repeticije može se jasno videti na primeru *matematičke indukcije*. Pri sumiranju elemenata niza $(q_1, ..., q_n)$ proverava se da li sekvenca

```
S:=0;
for i:=1 to n do S:=S+q[i];
```

služi za računanje izraza $S = \sum_{i=1}^{n} q[i]$. Matematička indukcija svoju primenu zasniva na broju elemenata niza, n. Na samom početku potrebno je dokazati korektnost sekvence za vrednost n=1. Simboličkim izvršavanjem za vrednost n=1 jednostavno se može pokazati da je na kraju sekvence S=q[1]. Dalje, induktivna pretpostavka je da je sekvenca korektna za n=K-1, K>1. Na osnovu svega gore navedenog može se dokazati da osobina korektnosti važi za n=K. Ekvivalentan oblik sekvence se dobija transformacijom

```
S:=0;
for i:=1 to K-1 do S:=S+q[i];
S:=S+q[K];
```

Na bazi pretpostavke zaključujemo da prve dve naredbe za izlaz daju

$$S = \sum_{i=1}^{K-1} q[i]$$

.

Kao rezultat primene poslednje naredbe dobija se

$$S = \sum_{i=1}^{K-1} q[i] + q[K] = \sum_{i=1}^{K} q[i]$$

što je svakako i bio cilj dokazivanja.

3. ZAKLJUČAK

Programi se mogu testirati u raznim fazama razvoja i različitim stepenom strogosti. Kao i svaki deo razvoja testiranje zahteva rad. Testiranje programa je jedna od najskupljih pa, prema tome, i najvažnijih aktivnosti u toku njegovog životnog ciklusa. Sprovodi se, kako u toku prvobitne realizacije tako i u fazi eksploatacije i održavanja i to prilikom svake modifikacije programa. Cilj konstruktivnog testiranja je da pruži dokaz da nema greška što se svakako kosi sa ciljem destruktivnog pristupa testiranju. Metoda simboličkog izvršavanja programa je veoma dobra za testiranje manjih programa. U poslednje vreme ulaže se dosta napora da se konstruktivni pristup primeni parcijalno, tj. na određene delove programa, ako već ne može u celini. Testiranjem se utvrđuje prvo u kojoj meri program obavlja posao za koji je namenjen, a zatim i kako se ponaša u različitim eksploatacionim uslovima. Kod testiranja softvera najbitnije je odrediti niz test stavki za ono što se testira. Pre toga moramo razjasniti koje informacije se moraju nalaziti u test stavki. Najočitija informacija su ulazi kojih ima dve vrste: preduslovi (okolnosti koje postoje pre isvršenja test stavke) i stvarni ulazi, koje identifikujemo nekim metodom testiranja. Nema smisla testirati greške koje verovatno ne postoje. Mnogo je efikasnije dobro razmisliti o vrstama grešaka koje su najverovatnije (ili najštetnije) i tada odabrati metode testiranja koji će verovatno moći da otkriju takve greške.

LITERATURA

[1] Beizer B. “Software Testing techniques, Van Nostrand Reinhold, New York,1900.

[2] E. J. Weyuker and B. Jeng, Analyzing partition testing strategies, IEEE Transactions of Software Engineering, 1989

[3] Jorg P. *Software testing* -Paul C. Jorgensen, 1995

[4] Hoare C.A.R, Wirth N., *An axiomatic definition of the programming language Pascal*, Acta Informatica 2, 1995.

[5] J. Vass, *Testing software*, IEEE Transactions of Software Engineering

[6] Berkovic I, Markoski B., Setrjčić J., Brtka V., Dobrilovic D.. "Testing of program correctnes in formal theory", Ubiquitous Computing and Communication Journal Octobar 2009, ISSN Online 1992-8424 , Special Issue on ICIT 2009 conference - Bioinformatics and Image , Bioinformatics and Image , 7/30/2009

[7] Markoski B., Hotomski P., Malbaski D., "Verifying program corectness resolution metod", ETAI 2000, V National conference with international participation, Ohrid, FR Macedonia.

[8] [IEEE,1991] D. Ada, *Software testing and software development lifecycles*, IEEE Transactions of Software Engineering

[9] B. Pettichord, *Automated software testing*, IEEE Transactions of Software Engineering, 1989

[10] J. Menrigan, *Automated distributed testing*, IEEE Transactions of Software Engineering, 1990

[11] R.S. Freedman, *Testability of Software Components*, A publication of the IEEE computer society, 1991

[12] V.R. Basili, R.W. Selby, *Comparing the Effectiveness of Software Testing Strategies*, Components, A publication of the IEEE computer society, 1987

[13] Mantos A. Vasislis C. Kosti D, Systematicaly Testing A Real- Time Operating system. IEEE Transactions of Software Engineering, 1995

[14] Markoski B., Vasiljevic P., Babic Đ., "Definisanje plana testiranja programa", 10th International Conference Dependability and quality management ICDQM -2007, Belgrade , Serbia 2007

[15] Markoski B., Hotomski P.,Malbaski D., Bogićević N. "Testing the integration and the system", International ZEMAK symposium, Ohrid, FR Macedonia, 2004