# Metamorfno testiranje

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Rastko Đorđević, 1091/2017
rastko_djordjevic@matf.bg.ac.rs

12. decembar 2018

**Sažetak**

Metamorfno testiranje (eng. metamorphic testing) je tehnika testiranja u verifikaciji softvera koja se oslanja na osobine algoritma koji se testira. Omogućava testerima da u potpunosti automatizuju proces generisanja dodatnih test primera, u čemu i leži moć ove tehnike. U velikom broju oblasti nije moguće koristiti klasične metode verifikacije ili je njihova primena veoma skupa. U ovim slučajevima metamorfno testiranje znatno može doprineti testiranju aplikacije. Iako još u povoju, uspešno je korišćeno za testiranje poznatih aplikacija koje ogroman broj korisnika koristi svakodnevno.

## Sadržaj

# 1 Uvod

Proročište (eng. oracle) u verifikaciji softvera je procedura kojom testeri mogu da odluče da li je izlaz testiranog programa ispravan. Međutim u nekim situacijama proročište nije dostupno ili ga je previše teško iskoristiti. Ovo se naziva problem proročišta. Problem proročišta je izuzetno izražen u nekim oblastima, kao što su računarska grafika, konstrukcija kompilatora, mašinsko učenje. U nekim drugim situacijama, proročište je često ljudski tester, koji proverava rezultate testiranja manualno. Ovaj vid testiranja značajno smanjuje efikasnost i povećava troškove samog testiranja.

Metamorfno testiranje je metod koji testira programe bez korišćenja proročišta. Umesto toga koriste se osobine algoritma koji se testira kako bi se generisali dodatni test primeri i automatski verifikovali njihovi izlazi. U ostatku rada biće predstavljeni koncepti metamorfnog testiranja uz primere njegovih primena.

# 2 Metamorfno testiranje

U ovom poglavlju biće opisane osnove metamorfnog testiranja kao i neki praktični saveti za njegovo uspešno korišćenje.

## 2.1 Osnove metamorfnog testiranja

Problem proročišta je decenijama jedan od najvećih problema u testiranju softvera i bilo je nekoliko pokušaja da se on reši. Jedno od rešenja je da se koristi pseudo proročište [6], gde se koristi više implementacija algoritma čiji se izlazi nad istim test primerima porede. Ako su rezultati različiti onda bar jedan od algoritama ima grešku. Ova tehnika se često ne može primeniti pošto možda ne postoji više implementacija algoritama, ili ako postoji možda su kreirani od strane istih ljudi koji često prave iste greške. [4].

Međutim, čak i sa samo jednom implementacijom, većina aplikacija ima neka svojstva takva da za određene promene ulaza mogu da predvide neke karakteristike novog izlaza, uz poznavanje prvobitnog izlaza. Ovo je osnova metamorfnog testiranja koje je uvedeno u radu [2]. Formalniji opis ove tehnike glasi:

Za dati program $P$ i test primer $\bar{x}$, odgovarajući izlaz označen je sa $P(\bar{x})$. Pretpostavimo da je $\bar{x_0}$ test primer sa izlazom $P(\bar{x_0})$. Metamorfno testiranje podrazumeva konstruisanje velikog broja novih test primera $\bar{x_1}$, $\bar{x_2}$, ..., $\bar{x_k}$ na osnovu para test primera i izlaza ($\bar{x_0}$, $P(\bar{x_0})$) i grešaka koje se mogu povezati sa P. Ovi novi test primeri su dizajnirani da otkriju greške koje uspevaju da budu neotkrivene test primerom $\bar{x_0}$.

Metamorfno testiranje se u praksi može implementirati lako. Najteži deo je identifikovati sve metamorfne relacije (eng. metamorphic relations) koje daju neku vezu između više ulaza i njihovih izlaza za dati program. Potom se "izvorni" i "naknadni" test primeri mogu generisati na osnovu datih metamorfnih relacija. Potom je potrebno izvršiti sve test primere korišćenjem programa koji se testira i proveriti da li izlazi izvornih i naknadnih test primera zadovoljavaju odgovarajuću metamorfnu relaciju.

Metamorfno testiranje se treba primeniti zajedno sa ostalim strategijama selekcije test primera koje generišu inicijalni skup test primera.

## 2.2 Smernice za definisanje metamorfnih relacija

Opisana tehnika je bazirana na jednostavnom konceptu, nije teška za implementaciju, može se automatizovati i nezavisna je od određenog programskog jezika. Ključni korak pri ovoj tehnici je identifikacija metamorfnih relacija. Takođe je važno koje metamorfne relacije izabrati od ponuđenih. Za to postoji nekoliko principa koji se mogu poštovati, iz perspektive testiranja metodama bele kutije ali takođe i testiranja metodama crne kutije. Detaljan opis ovih principa može se naći u radu [1].

Najznačajniji je princip različitosti u izvršnim putevima, koji tvrdi da bi trebalo izabrati metamorfne relacije koje rezultuju najvećim razlikama u izvršnim putevima između izvornih i naknadnih test primera. Intuicija iza ovog razmišljanja je da ako imamo veliku razliku između izvršnih puteva, imamo i veliki prostor u kojem može da se izrazi propust u softveru. Ponekada nije očigledno koje metamorfne relacije će rezultovati većoj razlici između izvršnih puteva, tako da je u tim slučajevima neophodno pokrenuti program i analizirati pokrivenost koda za sve test primere.

Postoji još osobina koje mogu uticati na uspešnost određene metamorfne relacije u detekciji bagova. Bitna osobina odnosi se na tip relacije između relevantnih izlaza. Relacije ekvivalencije su pokazane kao bolje od ostalih relacija, pošto je ekvivalentnost uži uslov od ne-ekvivalentnih uslova. Kao posledica, metamorfne relacije sa relacijom ekvivalencije se lakše mogu prekršiti od ostalih relacija, čime bi trebalo da detektuju veći broj grešaka.

# 3 Primene

Razmatrana tehnika testiranja se može primenti kako na numeričke, tako i na ne-numeričke probleme. U nastavku poglavlja biće prikazane primene metamorfnog testiranja u raznim oblastima.

## 3.1 Računanje standardne devijacije niza brojeva

Jednostavan primer metamorfnog testiranja može biti nad programom koji računa standardnu devijaciju nad nizom brojeva. Određene transformacije niza bi trebalo da daju isti rezultat. Permutacija reda elemenata ne bi trebalo da utiče na konačni rezultat, kao ni množenje svake vrednosti sa -1, pošto devijacija od srednje vrednosti u oba slučaja ostaje ista.

Takođe postoje transformacije koje menjaju izlaz ali na očekivan način. Na primer, ako se svaki broj pomnoži sa nekom konstantom, standardna devijacija tog novog niza brojeva bi trebalo da bude srazmerno veća od standardne devijacije originalnog niza.

Stoga, za dat skup brojeva(izvorni test primer), metamorfnim relacijama koje su navedene mogu se generisati novi naknadni skupovi brojeva (naknadni test primeri). Nakon računanja standardne devijacije svih pomenutih test primera, njihovim poređenjem se može testirati data aplikacija.

## 3.2 Najkraći put u grafu

Za dati težinski neusmereni graf $G$, početni čvor $x$ i ciljni čvor $y$ koji se nalaze u grafu $G$, treba vratiti put sa najmanjom distancom od $x$ do $y$.

Put koji je vraćen mora biti u obliku $x, v_1, v_2, ..., v_k, y$ i odgovarajuća distanca $p$ koja predstavlja zbir svih ivica na putu od $x$ do $y$. Krajnje

je netrivijalno proveriti da li je vraćeni put zaista minimalan za velike grafove. Jedna trivijalna provera bila bi da se proveri da li sve grane $(x, v_1)$, $(v_1, v_2)$, ..., $(v_k, y)$ pripadaju grafu G.

Pošto je u ovom slučaju u praksi neizvodljivo imati test oracle, metamorfno testiranje nam može značajno pomoći. Ako se pretraga vrši na neusmerenom grafu i ako je put od $x$ do $y$ minimalan, onda će biti minimalan i put od $y$ do $x$, tako da to možemo testirati. Za neusmerene grafove možemo iskoristiti sličan dizajn generisanja test primera. Naime, čvor $v_i$ koji pripada vraćenom minimalnom putu može biti iskorišćen tako da se izračuna minimalni put od početnog čvora $x$ do čvora $v_i$, kao i minimalni put od čvora $v_i$ do ciljnog čvora $y$. Sada se može proveriti da li je zbir distanci ova dva puta jednak distanci minimalnog puta između čvorova $x$ i $y$. Ovo se može testirati za sve čvorove koji pripadaju prvobitno vraćenom minimalnom putu.

## 3.3 Računarska grafika

Programe koji kao izlaz imaju ogromnu količinu podataka je izuzetno skupo verifikovati. Na primer, softver u računarskoj grafici generiše slike koje iscrtava na ekran. Međutim praktično je nemoguće da tester manuelno proveri da li je svaki piksel adekvatno iscrtan. U ovakvim situacijama praktičan pristup je da se proveri korektnost na određenom broju individualnih izlaza. Potom se nad ovim izlazima može primeniti metamorfno testiranje koje služi da testira izlaze većeg broja slučajeva.

Na primer, može se testirati kako radi osvetljenost u nekom programu. Prvo se generišu izvorni test primeri sa pozicijom svetla na određenom mestu. Nakon toga treba uvideti da važi metamorfna relacija koja tvrdi da ako se izvor svetlosti pomeri, osvetljenost objekata takođe treba da se promeni u skladu sa odgovarajućim formulama. Ovo je jednostavan način da se automatski proveri da li se pikseli iscrtavaju korektno.

Ovo je samo jedan od mnogobrojnih primera metamorfnih relacija koje mogu biti od koristi u računarskoj grafici.

## 3.4 Konstrukcija kompilatora

Testiranje kompilatora je poznato po svojoj težini. Ovo je posledica toga što je teško verifikovati ekvivalenciju između izvornog koda i objektnog koda. U nastavku će biti dat primer koji ilustruje kako metamorfno testiranje može olakšati ovaj problem.

Za dati izvorni kod, mogu se dodati određene linije koje mu ne menjaju semantiku. Na primer možemo opkružiti neki izraz if uslovom, koji uvek prolazi:

```
if ( 1 == 1 ){
        izraz
}
```

Takođe mogu se izmeniti linije izvornog koda sa semantički ekvivalentnim linijama koda zapisanim na drugi način. Primer ove izmene je da se izraz $x = y * z$ može zameniti izrazom $x = y * z + 0$. Potom svi generisani izvorni kodovi mogu biti propušteni kroz kompilator. Njihovi izlazi bi trebalo da se ponašaju na isti način budući da su na izvornom kodu izvršene promene koje čuvaju semantiku programa.

### 3.5 Mašinsko učenje

U radu [7] objavljenom 2011. godine predložena je primena metamorfnog testiranja na klasifikatore u mašinskom učenju. Oni su vršili testiranja na poznatoj Weka kolekciji algoritama. Bazirali su se na implementacije k najbližih suseda i naivnog bajesovskog klasifikatora. Uspeli su da pronađu propuste u implementacijama oba klasifikatora.

Takođe su uporedili performanse metamorfnog testiranja sa unakrsnom validacijom, koja se veoma često koristi u mašinskom učenju. Ubacivali su greške u implementacije algoritama i testirali su koja tehnika će detektovati koji vid grešaka. Pokazano je da su neke česte programerske greške, koje je teško detektovati unakrsnom validacijom, detektovane korišćenjem metamorfnog testiranja.

Neke od intuitivnih metamorfnih relacija koje su koristili su:

1. Konzistentnost sa afinim transformacijama
2. Permutacija labela klase
3. Permutacija atributa
4. Dodatak neinformativnih atributa
5. Dodatak informativnih atributa
6. Konzistentnost sa ponovnom predikcijom
7. Dodatni trening podaci

## 4 Zaključak

U radu su navedeni principi za korišćenje metamorfnog testiranja u različitim oblastima. Posle 20 godina od publikacije prvog rada o ovoj temi, metamorfno testiranje je pokazalo veliki broj obećavajućih rezultata i detekcija velikog broja propusta u popularnim aplikacijama kao što su GCC i LLVM [5]. Ove uspešne primene kao i mnoge druge su sakupljene i opisane u nedavno objavljenom radu [3]

Jedinstvenost metamorfnog testiranja je u tome što ne zahteva ljudsko angažovanje za generisanje naknadnih test primera. Stoga u potpunosti omogućava automatizaciju procesa testiranja. Takođe se može kombinovati sa bilo kojim strategijama za selekciju test primera.

Važno je napomenuti da metamorfno testiranje nije rešenje svih problema. Potrebno je dobro domensko poznavanje problema kako bi primena ovog vida testiranja bila efektivna. Takođe, pošto metamorfno testiranje proverava samo neophodne uslove i pošto ne testira korektnost individualnih izlaza, neophodno je koristiti ga uz neki drugi vid testiranja kako bi se ustanovila pouzdanost u korektnost programa.

U budućnosti ima puno prostora za napredak oblasti metamorfnog testiranja, koja iako obećavajuća još uvek nije iskorišćena u skladu sa svojim punim potencijalom.

## Literatura

[1] T. Y. Chen, D. H. Huang, T. H. Tse, and Zhi Quan Zhou. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, pages 569–583, 2004.

[2] Tsong Yueh Chen, Samson Cheung, Siu Ming Yiu, Hong Kong Research, Grant Council, and Corresponding Author. Metamorphic testing: a new approach for generating next test cases.

[3] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Comput. Surv.*, 51(1):4:1–4:27, January 2018.

[4] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, SE-12(1):96–109, Jan 1986.

[5] Vu Le, Mehrdad Afshari, and Zhendong Su. Compiler validation via equivalence modulo inputs. *SIGPLAN Not.*, 49(6):216–226, June 2014.

[6] Elaine J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[7] Xiaoyuan Xie, Joshua W.K. Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544 – 558, 2011. The Ninth International Conference on Quality Software.