

Invarijante

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Jovan Dmitrović, 1094/2018
jdmitrovic@gmail.com

15. januar 2020

Sažetak

Ovaj rad opisuje šta su to, zapravo, invarijante, koliki je domen njihove upotrebe i koliko su bitne u oblasti verifikacije softvera. Pored definisanja samog pojma invarijante, biće prikazani i primeri problema koji se mogu olakšati korišćenjem invarijanti. Na kraju će biti prikazan i *Daikon*, alat koji automatski pronalazi invarijante u kôdu.

Sadržaj

1	Uvod	2
2	Primeri invarijanti u matematici	2
2.1	MU zagonetka	2
3	Invarijante u programiranju	3
3.1	Invarijante petlje	3
3.2	Invarijante klase	3
4	Dinamičko pronalaženje invarijanti	4
4.1	Daikon	4
4.1.1	Primer korišćenja alata Daikon	4
5	Zaključak	5
	Literatura	5

1 Uvod

Invarijanta[9] predstavlja svojstvo matematičkih objekata koje se ne menja prilikom korišćenja određenih klasa transformacija. Formalnija definicija bi podrazumevala da su invarijante tačne za sve objekte u istoj klasi u odnosu na neku relaciju ekvivalencije[8].

Pokazuje se da su invarijante korisne u mnogim granama matematike kao što su geometrija, algebra ili diskretna matematika. Pored matematike, invarijante su bitne i u informatici, tačnije, u verifikaciji softvera, što će biti opisano u nastavku teksta.

2 Primeri invarijanti u matematici

Jedan od osnovnih primera invarijanti jeste da se uglovi i razmere ne menjaju prilikom rotacije, simetrije, skaliranja ili translacije u Euklidskom prostoru; ova činjenica je osnova trigonometrije. Međutim, uglovi i razmere nisu invarijantne u odnosu na neke druge transformacije kao što je, na primer, smicanje.

Neki od kompleksnijih primera invarijanti u matematici su:

- Sopstvene vrednosti matrice su invarijantne u odnosu na ortogonalne transformacije.
- Varijansa slučajne promenljive je invarijantna u odnosu na dodavanje konstante na sve vrednosti te slučajne promenljive.
- *Lebegova mera*[5] je invarijantna u odnosu na translaciju.

2.1 MU zagonetka

U knjizi "Gedel, Ešer, Bah: Beskrajna zlatna nit", za koju je njen autor, Daglas Hofšteter, dobio Pulicerovu nagradu 1980. godine, našao se problem tzv. *MU zagonetke*[4]. MU zagonetka je zadata u okviru jednostavnog formalnog sistema koji se naziva "MIU".

MIU sistem čine niske koje obrazuju karakteri M, I i U. Zadatak je, polazeći od početne (aksiomatske) niske MI doći do niske MU koristeći sledeća pravila transformacija, podrazumevajući da x i y predstavljaju proizvoljne niske sistema MIU:

1. $(xI \rightarrow xIU)$ Na kraju niske x koja se završava sa I može se dodati karakter U.
2. $(Mx \rightarrow Mxx)$ Ako niska počinje sa karakterom M, niska nakon M se može duplirati.
3. $(xIIIy \rightarrow xUy)$ Ukoliko niska sadrži tri uzastopna karaktera I, ti karakteri se mogu zameniti jednim karakterom U.
4. $(xUUy \rightarrow xy)$ Ukoliko niska sadrži dva uzastopna pojavljivanja karaktera U, oni se mogu obrisati.

Posmatrajmo broj pojavljivanja karaktera I (označimo ga sa n) u nisci nakon korišćenja svakog pojedinačnog pravila transformacije: prilikom korišćenja prvog i četvrtog pravila, n se ne menja. Nakon korišćenja drugog pravila dobija se $2n$ pojavljivanja, dok se nakon korišćenja trećeg pravila dobija $n - 3$ pojavljivanja karaktera I.

Imajući prethodna opažanja u vidu, kao i činjenicu da krećemo od niske MI, može se zaključiti da broj pojavljivanja karaktera I u nisci nije

deljiv sa 3, bez obzira kakva transformacija je korišćena. Drugim rečima, jednačina modularne aritmetike $n \not\equiv 0 \pmod{3}$ je *invarijanta* sistema MIU u odnosu na data pravila transformisanja niski. Samim tim, važi da $n \neq 0$, time dokazujući da je nemoguće doći do niske MU.

3 Invarijante u programiranju

Invarijante u softveru se uglavnom koriste u svrhe implicitne verifikacije softvera; ukoliko napisani kod je napisan u odnosu na neku invarijantu, imamo garanciju da će kod uvek ispunjavati pravila koje ta invarijanta nameće.

Dve osnovne vrste invarijanti u softveru su:

- Invarijante petlje
- Invarijante klase

3.1 Invarijante petlje

Invarijanta petlje[7] (engl. *loop invariant*) je logička formula promenljivih u okviru koda koja je tačna pre i posle svake iteracije petlje. Tačnije, da bi neka logička formula bila invarijanta date petlje, mora da ispunjava sledeće uslove[6]:

1. Logička formula mora da bude tačna pre prvog ulaska u petlju.
2. Logička formula mora da bude tačna nakon svakog izvršavanja tela petlje.

Uzmimo sledeći primer, napisan u programskom jeziku C:

```
unsigned i = 0, p = 0;
while(i < x){
    p = p + y;
    i++;
}
```

Očigledno je da je u pitanju računanje proizvoda dva nenegativna broja korišćenjem sabiranja. Dokažimo da je logička formula $i \leq x \wedge p = y * i$ invarijanta date petlje:

1. Važi da je $i = 0$, pa je data logička formula je tačna, bez obzira na x i y .
2. Pretpostavimo da je logička formula tačna za neko i , dokažimo da je tačna i za $i' = i + 1$. Prvi konjunkt formule je uvek tačan zato što je u okviru uslova petlje, i važi da je $p = y * i$ u i -toj iteraciji. U sledećoj iteraciji, p se povećava za y , te važi da je $p' = p + y = y * i + y = y * (i + 1) = y * i'$.

Ovime smo dokazali traženo; iz ovog primera se može uočiti da je dokazivanje valjanosti invarijante petlje analogno dokazivanju indukcijom u matematici.

3.2 Invarijante klase

Invarijante klase su tip invarijanti prisutan u objektno-orijentisanom programiranju. One predstavljaju zakonitosti koje važe za svaki objekat određene klase, pre i posle korišćenja bilo kog konstruktora ili bilo koje funkcije date klase.

Da bi objekti poštovali pravila koje invarijanta nameće, moguće je koristiti tzv. *assert* funkcionalnost, koja postoji u većini modernih programskih jezika kao što su, na primer, Python i C++. Ona omogućava konstruktorima i funkcijama da izbacuju izuzetke kada neki određeni uslov (u ovom slučaju invarijanta) nije ispunjen. Pored *assert*-a, neki programski jezici (npr. Eiffel[3] i D[1]) podržavaju i nativno pisanje klasnih invarijanti direktno u kôd.

4 Dinamičko pronalaženje invarijanti

Pronalaženje invarijanti "ručno" pokazuje se neisplativim; time se troši velika količina dragocenog vremena programera. Zbog toga je interesantan problem nalaženja invarijanti u softveru dinamički, odnosno, nalaženje invarijanti u kôdu pomoću specijalizovanog softvera.

4.1 Daikon

Jedan od programa korišćenih u svrhu dinamičkog pronalaženja invarijanti jeste i *Daikon*[2], koji je nastao na Univerzitetu u Vašingtonu 1998. godine. Način na koji Daikon funkcioniše jeste da se prvo dobija instrumentalizovana verzija izvornog kôda korišćenjem određenog *front-end* alata (*Chicory* i *Kvasir*, alati za programski jezik Java, odnosno C++ su uključeni u instalaciju samog Daikon-a). Instrumentalizacijom se prate određene promenljive u izvornom kôdu; nakon toga, pokreće se prethodno napisan skup testova nad praćenim promenljivama, i na kraju se pronalaze potencijalne invarijante. Čitav postupak, osim pisanja i odabiranja odgovarajućeg skupa testova, se izvršava automatski.

Bitno je napomenuti da Daikon ne garantuje korektnost niti kompletnost pronađenih invarijanti; to jest, pronađene invarijante ne moraju biti tačne, niti su sve moguće invarijante pronađene.

4.1.1 Primer korišćenja alata Daikon

Recimo da imamo program napisan u programskom jeziku C. Prvo je neophodno se pozicionirati u direktorijum gde se izvorni kôd programa nalazi. Nakon toga, potrebno je kompajlirati kôd sa dodatnim *debug* informacijama (sa komandom `-gdwarf-2`) i bez dodatnih optimizacija:

```
gcc -gdwarf-2 -no-pie primer.c -o primer
```

Sada kada imamo izvršni fajl, koristimo *Kvasir*, *front-end* Daikon-a za C/C++ programe tako što pokrenemo `kvasir-dtrace` i u nastavku unosimo standardan način pokretanja programa u komandnoj liniji:

```
kvasir-dtrace ./primer prvi_argument drugi_argument
```

Izvršavanje ove komande pravi `daikon-output` direktorijum, u kom se nalaze fajlovi `primer.dtrace`, koji izlistava vrednosti promenljivih, i `primer.decl`, koji daje informacije o promenljivama i funkcijama koje se nalaze u programu, kao i o apstraktnim tipovima u koji se promenljive grupišu.

Pozicionirajmo se u direktorijum `daikon-output`. Na kraju, pokrenimo Daikon:

```
java -cp $DAIKONDIR/daikon.jar daikon.Daikon \  
    primer.dtrace primer.decl
```

Po završetku izvršavanja, pronađene invarijante biće smeštene u fajl `primer.inv.gz`.

5 Zaključak

Invarijante se pokazuju vrlo korisnim: one mogu pomagati pisanju kvalitetnijeg koda, pronalaženju bagova, pisanju dokumentacije, poboljšavanju skupova testova... Međutim, pronalaženje invarijanti u opštem slučaju može biti vrlo kompleksno. Da bi se takav problem zaobišao, može se koristiti alati kao što je *Daikon*. Problem koji nastaje korišćenjem takvih alata jeste u tome što nisu ni korektni niti kompletni.

Sa druge strane, čak i uprkos nedostacima alatima za pronalaženje invarijanti, pronađene invarijante se pokazuju korisnim u praksi. Stoga, napredak ovakvih alata biće vredno pratiti u budućnosti.

Literatura

- [1] D Official Specification. Online at: <https://dlang.org/spec/spec.html>.
- [2] Daikon Official Website. Online at: <https://plse.cs.washington.edu/daikon/>.
- [3] Eiffel Official Documentation. Online at: <https://www.eiffel.org/doc/eiffel/Eiffel>.
- [4] D. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1979.
- [5] H. Lebesgue. *Intégrale, longueur, aire*, 1936.
- [6] F. Marić. Programiranje II, beleške sa predavanja. Online at: <http://poincare.matf.bg.ac.rs/~filip/ppj-rg/p2.pdf>.
- [7] S. Nilsson. Loop invariants, explained. Online at: <https://yourbasic.org/algorithms/loop-invariants-explained>.
- [8] V. L. Popov. Invariant. Online at: <https://www.encyclopediaofmath.org/index.php/Invariant>.
- [9] Eric W. Weisstein. Invariant. Online at: <http://mathworld.wolfram.com/Invariant.html>.