Sinteza, merenje i predikcija

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Ozren Demonja, Filip Novović, Marko Crnobrnja
mi12319@alas.matf.bg.ac.rs, mi12318@alas.matf.bg.ac.rs, mi12024@alas.matf.bg.ac.rs

septembar 2018.

Sažetak

U ovom tekstu predstavljen je projekat čija je zamisao odrediti uticaj različitih ali ekvivalentnih struktura na resurse potrebne za automatsko dokazivanje funkcionalne ekvivalentnosti C programa. U tu svrhu upotrebljene su skripte koje prave manje transformacije na postojećem korpusu C kodova i zatim mere vreme potrebno CBCM alatu za proveravanje modela da dokaže njihovu ekvivalentnost. Konkretna implementacija pokazala se nedovoljnom za zadovoljavajuće predviđanje ovog vremena

Sadržaj

1 Uvod

Dokazivanje funkcionalne ekvivalencije između programa je korisna metoda verifikacije softvera. Ipak, ona zahteva znatnu količinu računarskih resursa. Kako bi se njihova upotreba mogla bolje organizovati, bilo bi poželjno odrediti neke zavisnosti između različitosti koje postoje između semantički ekvivalentnih programa i resursa potrebnih za dokaz ekvivalentnosti.

Ovaj rad će ispitati uticaj nekih jednostavnih transfomacija na vreme potrebno za dokaz ekvivalentnosti u alatu CBMC, pri različitim parametrima.

2 Problem

Kao jednostavan oblik različitosti ekvivalentnih kodova, razmatramo nekoliko transformacija programskog koda:

- Dodavanje koda koji nešto što ne utiče na ostatak programa.
- Prenošenje dela koda u funkciju.
- Pretvaranje for i while petlji jednih u druge, pretvaranje u ekvivalentnu konstrukciju sa goto naredbom.
- Pretvaranje množenja u iterativno sabiranje, deljenje u oduzimanje.
- Zamena operatora poređenja sa negacijom suprotnog operatora, zamena logičkih veznika po De Morganovim zakonima.
- Zamena inkrementiranih izraza sa ++ ili += ekvivalentnim naredbama dodele.
- Transformacija uslovnih naredbi u izraze sa ternarnim operatorom i obrnuto, zamena switch naredbe uslovnom naredbom.
- Transformacija break i continue naredbi sa ekvivalentnim goto naredbama.

Iako je daleko od toga da se razlika između svih ili čak većine ekvivalentnih programa može svesti na ovakve transformacije, one ipak predstavljaju ne potpuno trivijalne izmene čija svojstva prilikom dokazivanja ekvivalentnosti vredi ispitati.

Ove transformacije primenjene su programski na korpus rudimentarnih C programa i ekvivalentnost tako izmenjenih programa dokazana je u alatu CBMC.

3 Arhitektura sistema

Projekat je implementiran pomoću skriptova pisanih u programskom jeziku Pajton. Glavni moduli sistema implementiraju klasu koja obmotava kod, parser koji raščlanjuje kod, transformacije koda, grupisane po sličnosti; generisanje transformisanih programa i merenje vremena dokaza.

3.1 Omotač koda

Klasa *Code* implementirana je u datoteci **code_container.py**. Ova klasa čita programski kod sa lokacije date u konstruktoru i omogućava čitanje, spajanje i snimanje koda.

3.2 Parser

Klasa *Parser* implementirana je u datoteci **code_parser.py**. Njena funkcija je izdvajanje nezavisnih celina koda iz *Code* objekta koji joj je prosleđen. Implementiran je na osnovu regularnih izraza koristeći paket **re**. Pored izvlačenja traženih blokova naredbi, parser takođe pronalazi koje se naredbe koje se mogu transformisati se nalaze u programu i vraća njihovu listu.

3.3 Transformacije

Transformacije su implementirane u datotekama **add_function_transform.py**, **arithmetic_transform.py**, **branch_transform.py**, **loop_transform.py**, **relational_logical_transform.py** i pomoćnom modulu **transform.py**.

Ovi moduli su uglavnom zasnovani na funkcijama oblika *getXData*, *formY* i *XToYTransform*, gde prva uzima potrebne vrednosti iz naredbi, druga formira naredbe na osnovu naredbi a treća koristi prve dve za željenu transformaciju.

Razmotrimo na primer funkcije koje se koriste za transformaciju for petlje u while petlju u **loop_transform.py**:

```
def getForData(code):
    pattern = re.compile(r'(\w+)\s+.*;\s*\n*\s*for
        (.*);(.*);(.*){')
    for match in re.finditer(pattern, code):
        variable = match.group(1)
        init = match.group(2).strip()
        condition = match.group(3).strip()
        increment = match.group(4)[:-1].strip()
        statements = getStatementsInsideCurlyBraces(
            code)

        return (variable + "␣" + init[1:] + ";\n",
            condition, increment, statements)

#...

def formWhileLoop(variable, condition, increment,
    statements):
    loop = variable
    loop += "while(" + condition + ")"
    statements = addStatementInsideBlock(statements,
        "\n" + increment + ";\n", on\_begin=False)

    return loop + statements

#...

def forToWhileTransform(code):
    variable, condition, increment, statements =
        getForData(code)
    return formWhileLoop(variable, condition,
        increment, statements)
}
```

3.4 Generisanje programa

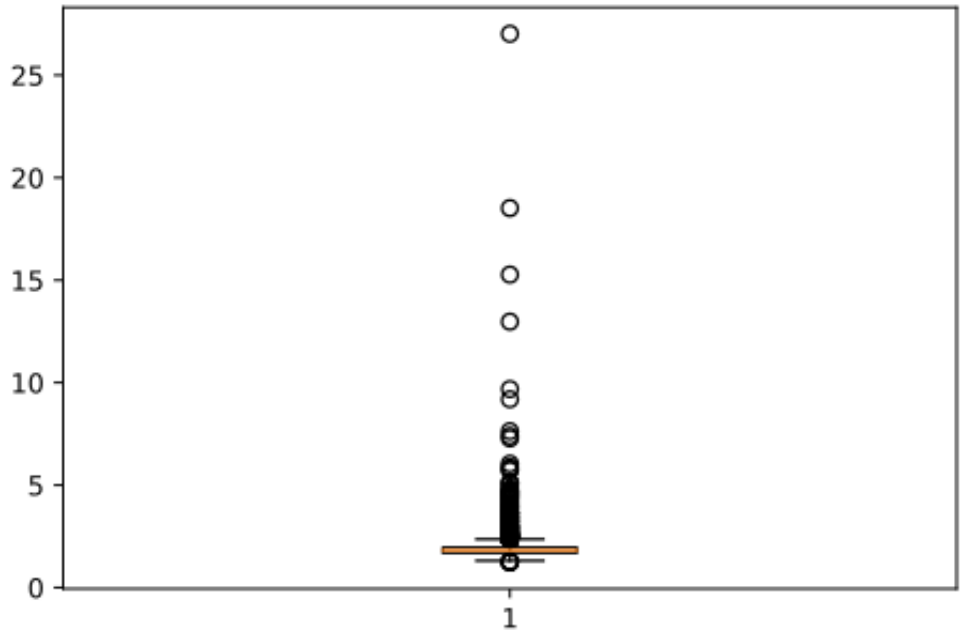
Upotreba transformacija nad datim korpusom jednostavnih programa koji se nalaze u direktorijumu **test code** implementirana je u **test.py**. Ona prolazi kroz listu mogućih transformacija za dati program koju dobija od parsera i zatim ih redom primenjuje i snima tako dobijene programe u direktorijum **function_equivalent_code**.

3.5 Merenje vremena

U modulu **measure.py** nalazi se funkcija *getCbmcOutputAndExecutionTime* koja pokreće CBMC kao potproces, meri vreme koje zahteva i vraća rezultat koji ovaj vraća. Pored toga, ovde se nalaze i funkcije za stvaranje i pisanje direktorijuma sa dobijenim podacima. U **measure_test.py** je kod koji koristi ovaj modul da generiše potrebne podatke za sve primere u **function_equivalent_code**.

4 Rezultati

Generisane podatke istražili smo koristeći Python skripte koje su sačuvane u obliku Jupyter sveske kao **Analysis.ipnyb**. Kako je postojao određen broj primera gde verifikacija nije uspela, oni su izbačeni iz daljeg razmatranja. Od dve mere trajanja verifikacije, *execution time* i *cbmc time* od kojih prva meri trajanje procesa a druga vreme koje CBMC prijavljuje, posmatrana je prva jer druga iz nepoznatih razloga nije prijavljena u svim slučajevima. Na Slici 1 prikazan je boksplot *execution time*.



Slika 1: Boksplot koji predstavlja raspodelu *execution time*

Kako bi se otkrila zavisnost vremena izvršavanja i primenjenih transformacija i parametara verifikacije, primenjena je grebena regresija sa unakrsnom validacijom sa prethodnom standardizacijom ciljane promenljive iz paketa **scikit-learn**. Na tabeli 1 prikazani su dobijeni koeficijenti regresije. Ovo se može tumačiti kao da svaka od pomenutih transformacija zahteva $1.87ms + koeficijent * 0.44ms$ dok zadate opcije dodaju po $koeficijent * 0.44ms$ na ovo vreme.

–unwind	0.096294
–partial-loops	0.029497
–no-unwinding-assertions	0.033565
multiplication operator	0.000000
divide operator	0.000000
less operator	-0.016916
lessEq operator	0.131142
greater operator	-0.108558
greaterEq operator	-0.092847
eq operator	-0.099545
neq operator	-0.097952
and operator	-0.018259
or operator	-0.039410
decrement operator	0.000000
increment operator	-0.125474
if statement	0.077742
? statement	-0.048127
switch statement	-0.054649
continue statement	0.000000
break statement	-0.054649
recursion block	-0.014385
garbage block	-0.046018
for-while	-0.012523
for-goto	0.000000
while-goto	0.000000

Tabela 1: Koeficijenti regresije

Kao R^2 greška regresije dobijena je izrazito niska vrednost od 0.017 što ukazuje da naš model objašnjava samo zanemarljivo mali deo varijanse. Stoga dobijeni model ne može dati dobar algoritam za predikciju trajanja provere ekvivalencije.

5 Moguća unapređenja

Pouzdan algoritam predikcije verovatno bi zahtevao dodatne podatke o obimu i strukturi razmatranog programa pored informacije o konkretnoj transformaciji, jer se čini da ovi aspekti doprinose većini varijanse između različitih trajanja.

Druga mana koja bi se mogla ispraviti je uporeba *execution time* merenja kao zamene za *cbmc time*. Prva verovatno varira više u zavisnosti od incidentalnih pojava tokom izvršavanja verifikacije što doprinosi većem šumu u podacima.

U daljem ispitivanju problema, verovatno bi bilo preporučljivo koristiti i složeniji parser za generisanje transformacija koji bi mogao obraditi i strukture koje se ne mogu opisati regularnim izrazima.