Opis sistema

MTK je aplikacija za metamorfno testiranje kompilatora, po čemu je i dobila naziv. Njen prvi značajan deo čini transformator, čiji je cilj da generiše semantički ekvivalentne programe od nekog ulaznog. Drugi deo su *Python* skriptovi koji automatizuju transformacije i proveravaju ih.

Opis problema

U okviru kursa „Konstrukcija kompilatora" napravljen je projekat koji podrazumeva skup transformacija koje mogu da se primene na odgovarajuće delove koda, ali tako da se semantika samog programa ne promeni, tj. program mora i dalje da vrati isti rezultat. Primer transformacije je zamena *while* petlje sa *for* petljom, odmotavanje petlje itd.

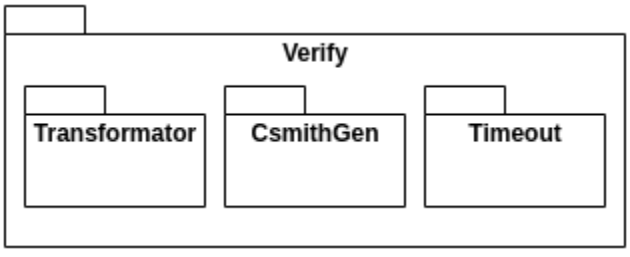
U okviru kursa „Verifikacija softvera" napravljena je nadogradnja projekta sa ciljem da se podigne pouzdanost i upotrebljivost transformacija. Takođe je obogaćen skup mogućih izmena. Ovako dobijene transformacije koriste se za potrebe metamorfnog testiranja kompilatora, gde se generiše veliki broj nasumičnih programa na koje se primenjuju transformacije i testira se da li rezultujući programi daju identičan rezultat, kao i da li se novodobijeni kod prevodi.

Opis arhitekture sistema

Pregled tehnologija:

- Transformacije su pisane u *C++*-u uz korišćenje *Clang* biblioteke.
- Za generisanje nasumičnih *C* programa koristi se *CSmith*. Generisani programi se postprocesiraju i verifikuju kroz *Python*.
- Metamorfno testiranje (proces koji objedinjuje sve delove) vrši se kroz *Python*.

Pogled kompozicije



Pogled kompozicije visokog nivoa apstrakcije

Timeout

Python modul koji omogućava pokretanje sporednih procesa sa postavljenim tajmerom koji prekida proces kada istekne vreme. Funkcije su specijalizovane za rešavanje problema

zombi procesa na *Linux* operativnom sistemu prilikom planiranog ili neplaniranog naglog prekidanja programa. Ovaj modul je nezavisno testiran i upotrebljiv u sličnim situacijama.

CSmithGen

Python modul koji služi za generisanje programa korišćenjem *CSmith* alata (okvir za alat). *Clang* biblioteka ne podržava rad sa celobrojnim tipovima fiksne širine u generisanim programima (npr. *int16_t*), zbog čega se ti tipovi u postprocesiranju zamenjuju tipovima sa kojim može da radi (npr. *long long*). Takođe, svi programi koji se generišu se završavaju u konačnom vremenu, tj. nijedan program nema beskonačnu petlju. Provera da li program sigurno nema beskonačnu petlju vrši se korišćenjem *Timeout* modula.

Transformator

Transformator klasa apstrahuje *cmake build* za biblioteku za transformacije i primenu tih transformacija nad generisanim *C* programima. Omogućava različite izbore načina primene transformacija (fleksibilna klasa) i provera da li semantika programa ostaje ista nakon primenjene sekvence transformacija. Smatra se da je program promenio semantiku ako važi jedno od sledećeg:

- Transformisani program daje drugačiji rezultat od polaznog.
- Transformisani program se ne prevodi.
- Transformisani program ima grešku pri izvršavanju.
- Transformisani program ima beskonačnu petlju (tačnije, izvršava se mnogo dugo u odnosu na inicijalni program).

Kada su u pitanju same implementirane transformacije, izdvaja se nekoliko grupa. Svaka izmena realizovana je kroz *Clang*-ov aplikativni programski interfejs prema apstraktnom sintaksnom stablu.

Prva implementirana transformacija je izmena petlji. Program transformiše sve petlje u C kodu u željeni tip – *for*, *while*, *do-while*. Za te potrebe koriste se klase *Do2ForVisitor*, *For2DoVisitor*, *For2WhileVisitor*, *PrepForVisitor*, *While2DoVisitor* i *While2ForVisitor*. Klasa *PrepForVisitor* priprema for petlje za dobru zamenu naredbe *continue*, dok ostale sprovode odgovarajuće transformacije. Tačne sheme izmena moguće je videti u svakom *cpp* fajlu.

Sledeća implementirana transformacija je odmotavanje petlji. Program odmotava (*unroll*, *unwind*) petlje u *C* kodu, umnožavajući im telo određeni broj puta. Za te potrebe koristi se klasa *LoopUnrollVisitor*, čija se shema rada može videti u odgovarajućem *cpp* fajlu.

Treća implementirana transformacija je izmena uslova. Program transformiše sve uslove u *C* kodu u željeni tip – *if-then-else*, *switch-case-default*. Za te potrebe koriste se klase *PrepIfVisitor*, *PrepIffVisitor*, *PrepSwitchVisitor*, *If2SwitchVisitor* i *Switch2IfVisitor*. Prve tri pripremaju *if* i *switch* naredbe za dobru zamenu naredbi *break* i *continue*, dok preostale sprovode odgovarajuće transformacije. Tačne sheme izmena moguće je videti u *cpp* fajlovima.

Pretposlednja implementirana transformacija je izmena rekurzije. Program transformiše svu kontrolu toka sa repnim ponavljanjem u C kodu u željeni tip – iteracija, rekurzija. Za te potrebe koriste se klase *Iter2RekVisitor*, *Rek2IterVisitor*, *FinIterVisitor* i *FinRekVisitor*. Prve dve

sprovode odgovarajuće transformacije, dok druge dve sprovode završne dorade, kako bi kod ostao sintaksno i značenjski ispravan. Tačne sheme izmena moguće je videti u *cpp* fajlovima.

Peta implementirana transformacija je imputacija koda. Program umeće (imputira) nove *AST* čvorove u *C* kod, menjajući pritom svaki *n*-ti već postojeći. Za te potrebe koristi se klasa *CodeImputVisitor*, pri čemu umetnuti kod ne menja semantiku polaznog programa, pošto se suštinski ne izvršava.

Poslednja šesta transformacija je eliminacija petlji. Petlje se eliminišu tako što se transformišu u semantički ekvivalentnu sekvencu labela, uslovnih i bezuslovnih skokova i samo tela petlji. Klase PrepWhile2GotoVisitor, PrepFor2GotoVisitor, PrepDo2GotoVisitor pripremaju telo petlje za transformaciju tako što menjaju break i continue naredbe sa skokovima na odgovarajuće labele petlje u kojima se nalaze. Zatim klase While2GotoVisitor, For2GotoVisitor i Do2GotoVisitor transformišu samu petlju u labele, uslovne i bezuslovne skokove.

Rad glavne (*main*) funkcije zasnovan je na pomoćnim klasama *MTKContext* i *MTKTransformer*, koje rekurzivno posećuju *AST* stablo polaznog programa.

Važno je utvrditi ispravnost samog transformatora, što je urađeno pisanjem sveobuhvatne svite testova, koja u suštini sprovodi integracione testove, odnosno proverava samo konačni rezultat.

U okviru klase *MTKTest* napisani su svi testovi. Redom je testirana svaka grupa transformacija: izmene i odmotavanje petlji, izmene uslova, izmene rekurzije, imputacija koda. Na kraju je proveren rad sa raznim tipovima neispravnog izlaza.

Primenjena je tehnika testiranja bele kutije, koja uzima u obzir različite putanje izvršavanja, pri čemu je postignuta faktički potpuna (~100%) pokrivenost linija koda i funkcija, kao i sasvim zadovoljavajuća visoka (~93%) pokrivenost grana.

Verify

Modul koji objedinjuje proces generisanja programa, transformaciju programa i verifikaciju transformisanih programa korišćenjem *CSmithGen* i *Transformator*.

Svaka iteracija podrazumeva generisanje *C* programa preko *CSmith* alata. Alat generise nasumični *C* kod (koji ne mora da se završava). Izlaz iz ovog programa je kontrolna suma (*checksum*) izračunata nizom nasumičnih operacija. Ukoliko se kod sigurno uspešno završava (ovo se proverava prevođenjem i pokretanjem), onda se izvršava niz transformacija nad kodom. Nakon toga se transformisani *C* kod prevodi i pokreće opet. Ukoliko se kod uspešno prevodi i daje identičan izlaz, onda se smatra da je test prošao. U suprotnom je test pao i čuva se u odgovarajućem direktorijumu.

Pregled zavisnosti

	Timeout	CsmithGen	Transformator	Verify
Timeout				

CsmithGen				
Transformator				
Verify				

Pogled aktivnosti