Opis sistema

Implementacija novih proveravača za Clang

Svetlana Bićanin, Lucija Miličić, Aleksandra Pešić

1 Opis problema

U okviru LLVM infrastrukture postoji statički analizator Clang, u okviru kog je potrebno implementirati neku novu proveru. Provere služe programerima da im daju upozorenje o potencijalnim greškama u kodu.

U okviru ovog projekta su implementirane dve nove provere, provera dodele u uslovima grananja i provera beskonačnih petlji.

1.1 Provera dodele u uslovima grananja

Potrebno je implementirati proveravač koji utvrđuje da li se u uslovu grananja ili petlje nalazi dodela vrednosti umesto izraza koji ima istinitosnu vrednost (True/False, 1/0). Pored toga, u slučaju složenih uslova (koji sadrže operator zarez), potrebno je još i proveriti da li izrazi koji prethode samom uslovu proizvode neki bočni efekat, što se od njih očekuje.

1.2 Provera beskonačnih petlji

Potrebno je proveriti da li se u kodu nalazi beskonacna petlja bez uslova zaustavljanja. Pošto ovaj problem nije moguće rešiti u opštem slučaju, potrebno je implementirati provere za neke specifične situacije u kojima treba prijaviti beskonačnu petlju.

2 Opis arhitekture sistema

Projekat LLVM sastoji se iz biblioteka i alata koji zajedno čine veliku kompajlersku infrastrukturu. Predstavlja skup modularnih i ponovno iskoristivih kompajlerskih tehnologija, čiji je cilj podrška statičkoj i dinamičkoj kompilaciji proizvoljnih programskih jezika.

Clang je kompilator otvorenog koda za C familiju jezika. Koristi LLVM optimizator i generator koda. Clang statički analizator je deo Clang projekta.

Clang statički analizator koristi razne implementacije proveravača (engl. checkers) prilikom analize. Proveravači su kategorisani u familije - podrazumevani i eksperimentalni (alpha). Podrazumevani proveravači izvršavanju bezbednosne provere, prate korišenje API funkcija, traže mrtav kod i ostale logičke greške.

Eksperimentalni (alpha) proveravači nisu podrazumevano uključeni pošto često daju lažne pozitivne rezultate. Proveravači implementirani u okviru ovog projekta pripadaju familiji eksperimentalnih proveravača.

3 Opis rešenja problema

Oba proveravača u okviru ovog projekta implementirana su u programskom jeziku C++, u odvojenim fajlovima.

3.1 Provera dodele u uslovima grananja

Klasa kojom je implementiran ovaj proveravač nasleđuje klasu Checker, što je i neophodno pri implementaciji bilo kog novog proveravača. U ovom slučaju nasleđuje konkretno klasu Checker tipa BranchCondition, koji služi za analizu uslova grananja.

Metod iz nasleđene klase koji ovaj proveravač treba da implementira je checkBranchCondition, koji kao jedan od argumenata prima baš naredbu koja predstavlja uslove u if, while, for ili ?: naredbama. Drugi argument ovog metoda je kontekst koji se koristi pri ispisivanju upozorenja.

Za dobijenu naredbu se prvo proverava da li predstavlja neku od binarnih operacija dodele, a u slučaju izraza čija se vrednost implicitno kastuje ili izraza u zagradama vrši se rekurzivna provera podizraza. Ukoliko je u pitanju slozena naredba, proverava se krajnji desni operand zareza, nakon čega se vrši dekompozicija i rekurzivno proverava ostatak. U tom slučaju za leve operande očekujemo da proizvode neki bočni efekat, a samo krajnji desni predstavlja uslov.

Radi smanjenja lažnih upozorenja, unarni operatori koji takođe vrše dodelu vrednosti (++, - -) se ignorišu, odnosno za njih se ne prijavljuje upozorenje. Razlog za ovo je taj što oni predstavljaju česte uslove u praksi, npr. while(x- -).

3.2 Provera beskonačnih petlji

Glavni pristupi problemu podrazumevaju pronalazak nedostižnih naredbi za prekidanje petlje (break ili return) ili nedostižnih naredbi neposredno nakon petlje, kao i pronalazak petlje ciji je uslov konstantan (uvek tačan), a koja ne sadrži naredbu za prekidanje.

Za implementaciju prvog dela ovog proveravača korišćen je deo koda iz UnreachableCodeChecker.cpp, proveravača koji pronalazi nedostižne delove koda. Pomocu funkcija tog proveravača omogućeno je pronalaženje nedostižne naredbe, a zatim je dodatno implementirana analiza kojom se utvrđuje da li je u pitanju neki od narednih slučajeva: nedostižna je naredba break, nedostižna je naredba return koja se nalazi unutar petlje ili je nedostižna neka naredba koja se nalazi neposredno nakon petlje (što ukazuje na to da se ta petlja ne završava).

Prve dve situacije analizirane su proverom kojoj klasi pripada nedostižna naredba, dok se poslednja situacija pronalazi iteracijom kroz osnovne blokove koji prethode nedostižnom i proverava se da li je neki od njih telo petlje.

Drugi deo ovog proveravača podrazumeva analizu blokova, u slučaju da je pronađen blok koji predstavlja petlju (While ili For) prvo se proverava da li je njen uslov konstantan i uvek tačan, a zatim se rekurzivno analizira sadržaj tela petlje u potrazi za nekom naredbom prekida.

Na kraju se ispisuje upozorenje čija je poruka u skladu sa pronadjenom situacijom.