

UNIVERZITET U BEOGRADU ELEKTROTEHNIČKI FAKULTET



Diplomski rad
**GENERISANJE INTERAKTIVNOG
GRAFA KONTROLE TOKA
ZA MIKROJAVA BAJTKOD**

mentor
prof. dr Dragan Bojić

student
Ksenija Bulatović

Beograd, januar 2024.

SADRŽAJ

SADRŽAJ	2
1. UVOD	3
2. PREGLED KORIŠĆENIH POJMOVA I TEHNOLOGIJA	4
2.1. MIKROJAVA	4
2.2. MIKROJAVA BAJTKOD.....	4
2.3. OSNOVNI BLOKOVI.....	4
2.4. GRAF KONTROLE TOKA	5
2.5. KORIŠĆENE TEHNOLOGIJE	5
2.5.1. Java 5	
2.5.2. Graphviz Dot jezik.....	6
2.5.3. Microjava SymbolTable.....	6
3. PREGLED APLIKACIJE I NAČIN KORIŠĆENJA	7
3.1. PODEŠAVANJA PARAMETARA PRI POKRETANJU APLIKACIJE.....	7
3.2. INTERAKTIVNI GRAF KONTROLE TOKA	8
3.3. KORIŠĆENJE TABELE SIMBOLA.....	12
4. REALIZACIJA REŠENJA	13
4.1. STRUKTURA REŠENJA.....	13
4.2. UČITAVANJE PARAMETARA	13
4.3. ČUVANJE I UČITAVANJE TABELE SIMBOLA (OPCIONO)	13
4.4. PARSIRANJE OBJKTOG FAJLA	15
4.5. KREIRANJE GRAFIČKE REPREZENTACIJE GRAFA	16
4.6. PRIKAZ KROZ INTERAKTIVNU APLIKACIJU.....	16
5. ZAKLJUČAK.....	17
6. LITERATURA.....	18

1. Uvod

Tema ovog diplomskog rada je implementacija interaktivne aplikacije za generisanje grafa kontrole toka mikroJava bajtkoda. Cilj rada je napraviti aplikaciju koja će omogućiti pregled osnovnih blokova pojedinačnih funkcija i toka kontrole.

U drugom poglavlju će biti definisani pojmovi Mikrojava, Mikrojava bajtkoda, osnovnih blokova, grafa kontrole toka, kao i naveden pregled tehnologija korišćenih za razvijanje aplikacije. Svaka tehnologija ima svoj rezime.

U trećem poglavlju će biti objašnjeno kako se koristi aplikacija, uz slike izgleda same aplikacije. Biće prikazane funkcionalnosti aplikacije i prikaz izgleda aplikacije tokom kretanja kroz istu.

U četvrtom poglavlju biće opisana struktura rešenja, u vidu implementacije logičkih celina. U njih spadaju opisi učitavanja parametara, čuvanja i učitavanja tabele simbola, parsiranja objektnog fajla, kreiranja grafičke reprezentacije grafa i prikaza kroz interaktivnu aplikaciju.

2. PREGLED KORIŠĆENIH POJMOVA I TEHNOLOGIJA

2.1. Mikrojava

Programski jezik Mikrojava se koristi u praktičnom delu kursa programski prevodioci 1 na Elektrotehničkom fakultetu u Beogradu. Mikrojava je slična Javi, ali je mnogo jednostavnija. Poput Javinih programa, Mikrojava programi se prevode u Mikrojava bajtkod.

2.2. Mikrojava bajtkod

MikroJava bajtkod je međukod za MikroJava virtuelnu mašinu, a ona je slična Java VM-u, ali ima znatno manje instrukcija. Neke instrukcije su takođe pojednostavljene. Detaljan opis instrukcija se može naći u specifikaciji Mikrojava virtuelne mašine.

2.3. Osnovni blokovi

Osnovni blok je najduži mogući niz od jedne ili više instrukcija koje se uvek izvršavaju sekvencijalno bez skoka na neki deo koda, osim na prvu instrukciju, i bez skoka iz koda, osim nakon poslednje instrukcije.

Osobine osnovnog bloka:

- Tok kontrole toka može ući u osnovni blok samo preko prve instrukcije unutar bloka, tj. ne postoje skokovi unutar osnovnog bloka.
- Kontrola će napustiti blok bez prekida ili grananje, osim nakon poslednje instrukcije.

Osnovni blokovi postaju čvorovi grafa kontrole toka, čije grane označavaju iz kog bloka je moguće ići u koji blok.

2.4. Graf kontrole toka

Graf kontrole toka (CFG) je grafička reprezentacija kontrole toka tj. izvršavanja programa. On je orijentisan ka procesima i prikazuje sve putanje koje mogu biti pređene tokom izvršavanja programa. U grafu toka kontrole osnovni blokovi predstavljaju čvorove grafa, a usmerene grane putanju toka kontrole. Od bloka B1 postoji orijentisana grana ka bloku B2, ako B2 neposredno sledi B1 u nekoj izvršnoj sekvenci, odnosno ako:

- postoji uslovan ili bezuslovan skok od poslednjeg iskaza B1 ka vodi B2, ili
- B2 neposredno sledi B1 u međukodu, a B1 nema na kraju bezuslovni skok.

2.5. Korišćene tehnologije

Za razvoj aplikacije korišćen je jezik Java (1.8), kao osnovna platforma. Pored toga, korišćen je Graphviz DOT alat za generisanje vizuelne reprezentacije grafa. Java je izabrana radi kompatibilnosti sa projektima koji se rade u okviru predmeta Programski prevodioci 1 na Elektrotehničkom fakultetu u Beogradu. Kombinacija Jave i DOT jezika pružila je solidan temelj za efikasnu realizaciju aplikacije, obezbeđujući stabilnost, performanse i vizuelnu preglednost.

2.5.1. Java

Java je objektno orijentisan programski jezik koji je dizajniran da bude platformski nezavisan, što znači da se isti kod može izvršavati na različitim platformama. On se često koristi za razvoj raznovrsnih aplikacija. Java je poznata po obilju standardnih biblioteka koje pružaju već gotove funkcionalnosti, olakšavajući programerima rad u različitim domenima. Neke od korišćenih biblioteka su:

- java io/nio – ulazni i izlazni tok podataka, serializaciju i fajl sisteme
- java util – standardne strukture podataka, regexi
- java awt + swing – grafički korisnički interfejs

2.5.2. *Graphviz Dot jezik*

Graphviz je skup programa za vizuelizaciju strukturalnih informacija u obliku dijagrama apstraktnih grafova i mreža. Automatsko crtanje grafova ima mnogo važnih primena u softverskom inženjeringu, dizajnu baza podataka i veba, mrežama, kao i u vizuelnim interfejsima za mnoge druge oblasti. Za implementaciju rada korišćen je DOT program koji generiše vizuelne reprezentacije usmerenih grafova.

2.5.3. *Microjava SymbolTable*

Symboltable biblioteka, implementirana na Elektrotehničkom fakultetu u Beogradu na predmetu Programski prevodioci 1, sadrži razne pomoćne klase za obradu informacija o simbolima unutar programa tokom analize i kompajliranja Mikrojava programa.

3. PREGLED APLIKACIJE I NAČIN KORIŠĆENJA

U ovom poglavlju, opisan je detaljan rad sistema i način korišćenja aplikacije.

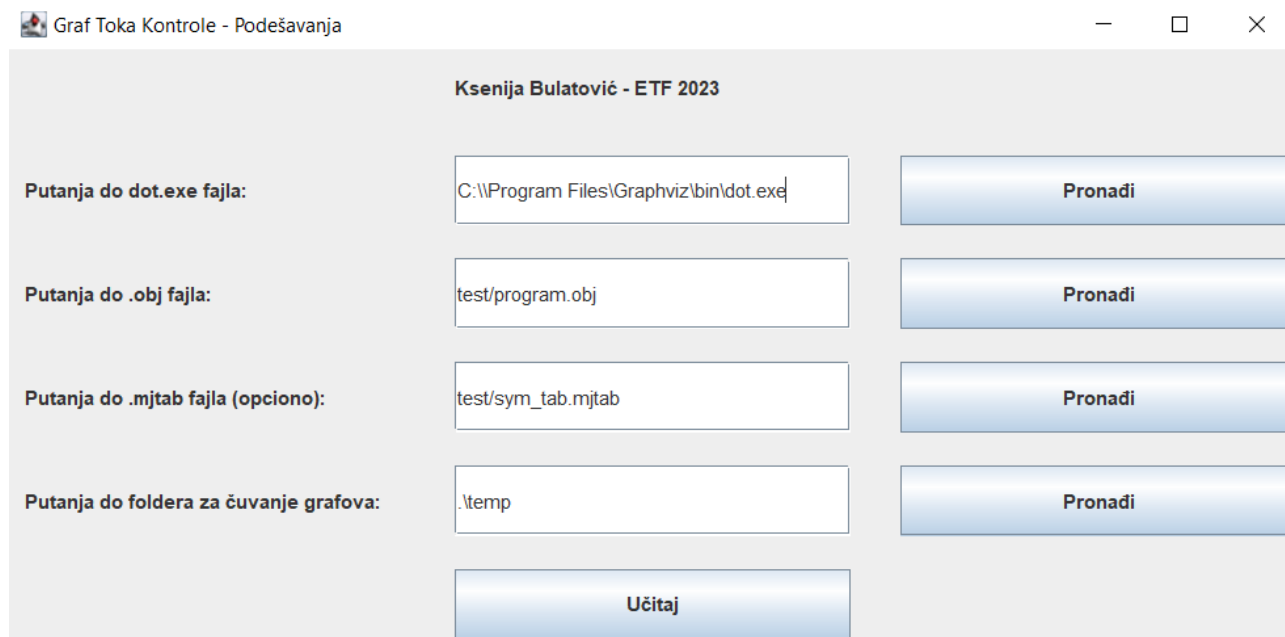
3.1. Podešavanja parametara pri pokretanju aplikacije

Pri pokretanju aplikacije (slika 3.1.1) potrebno je podesiti sledeće parametre za uspešno izvršavanje aplikacije:

- putanju do dot.exe fajla
- putanju do objektnog fajla koji sadrži mikroJava bajtkod
- putanju do fajla sa tabelom simbola, opciono
- putanju do foldera za čuvanje grafova

Radi lakšeg korišćenja postoje podrazumevane vrednosti kao na slici (slika 3.1.1). Svaku putanju je moguće promeniti pomoću „Pronađi“ dugmeta.

Klikom na dugme „Učitaj“ se zatvara otvoreni prozor i otvara novi prozor sa učitanim programom.



The screenshot shows a window titled "Graf Toka Kontrole - Podešavanja". Inside, the header reads "Ksenija Bulatović - ETF 2023". There are four rows of configuration fields, each with a label on the left, a text input field in the middle, and a "Pronađi" button on the right. The fields contain the following values: "C:\\Program Files\\Graphviz\\bin\\dot.exe", "test/program.obj", "test/sym_tab.mjtab", and ".\\temp". Below these fields is a large "Učitaj" button.

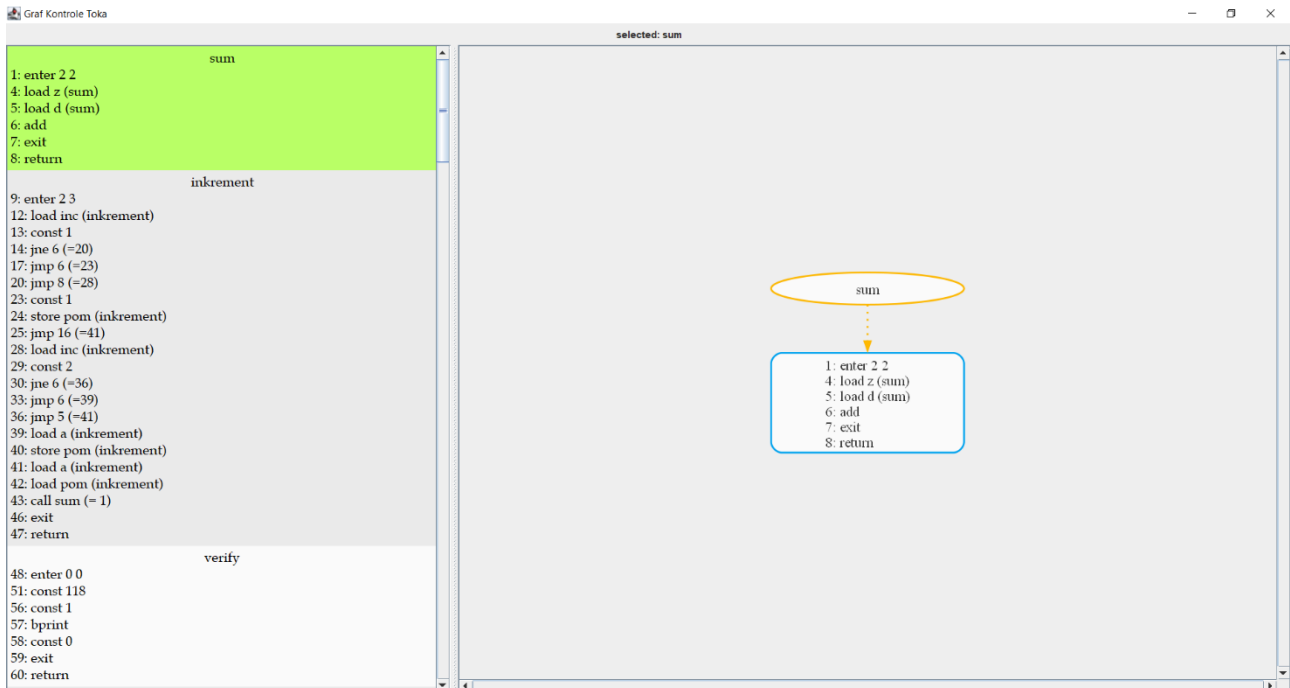
Label	Value	Action
Putanja do dot.exe fajla:	C:\\Program Files\\Graphviz\\bin\\dot.exe	Pronađi
Putanja do .obj fajla:	test/program.obj	Pronađi
Putanja do .mjtab fajla (opciono):	test/sym_tab.mjtab	Pronađi
Putanja do foldera za čuvanje grafova:	.\\temp	Pronađi

Učitaj

slika 3.1.1

3.2. Interaktivni graf kontrole toka

Nakon učitano programa izgled aplikacije je kao na slici (slika 3.2.1).



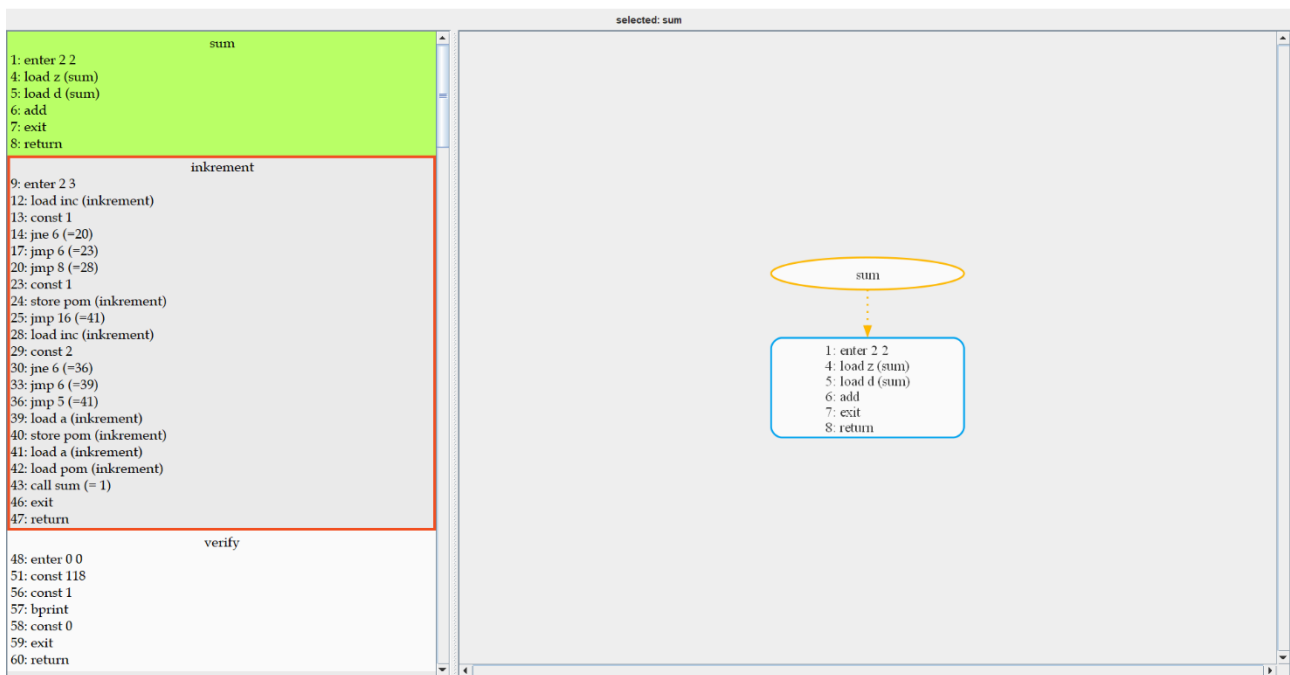
slika 3.2.1

Sa leve strane se nalazi mikroJava bajtkod koji je grupisan po funkcijama. Na vrhu svake funkcije, radi bolje preglednosti, nalazi se naziv funkcije.

Sa desne strane se nalazi graf kontrole toka za izabranu funkciju. Radi bolje preglednosti početni čvor elipsoidnog oblika žute boje sadrži naziv funkcije čiji se tok kontrole posmatra.

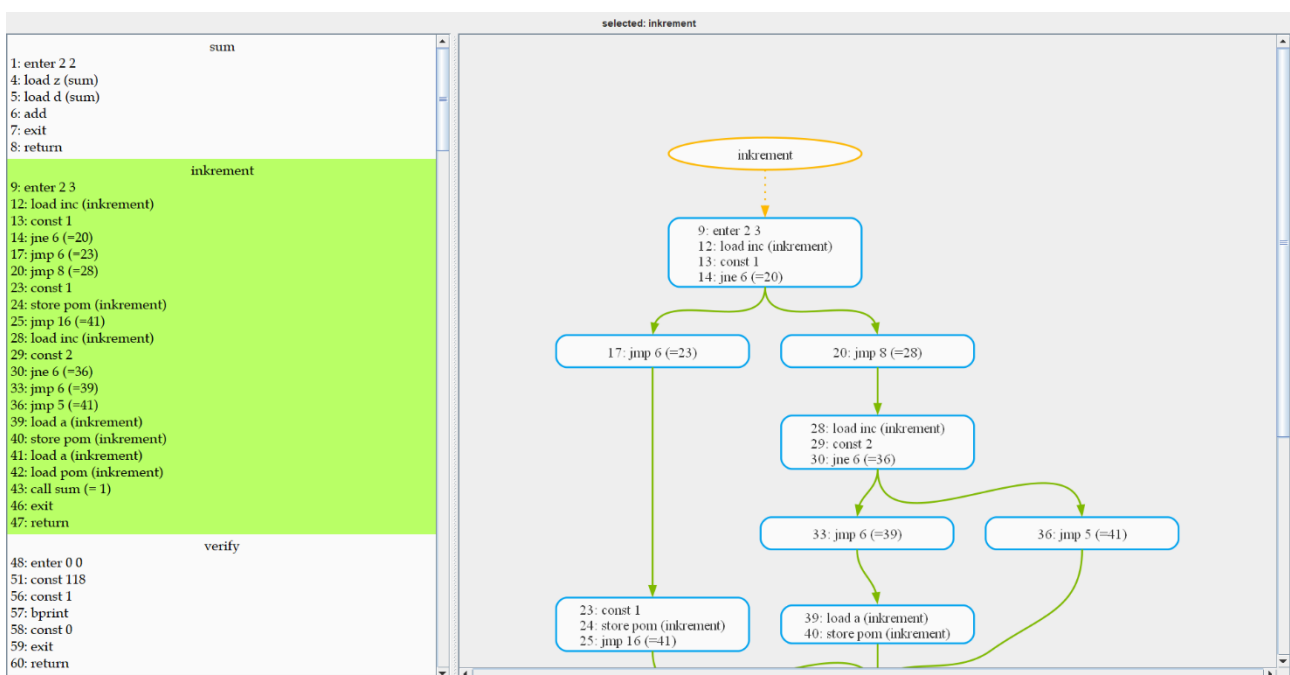
Na vrhu prozora je moguće videti naziv trenutno izabrane funkcije.

Prelaskom miša preko funkcija sa leve strane označeno je koja je funkcija u fokusu. (slika 3.2.2)



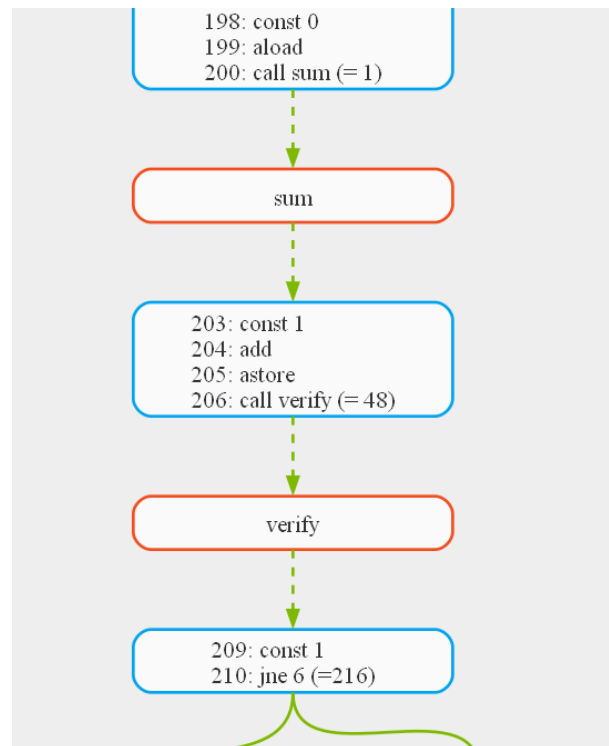
slika 3.2.2

Klikom na određenu funkciju sa leve strane, otvara se graf izabrane funkcije sa desne strane (slika 3.2.3)



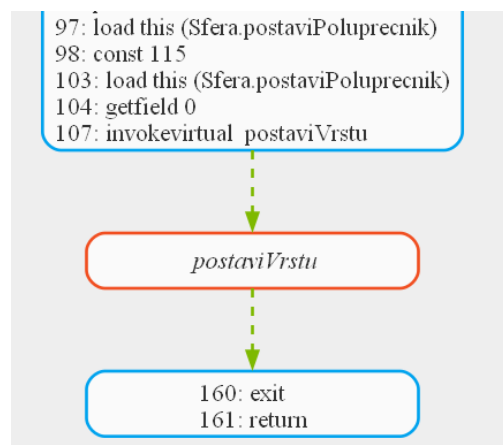
slika 3.2.3

Pozivi statičkih funkcija izgledaju kao na slici 3.2.4.



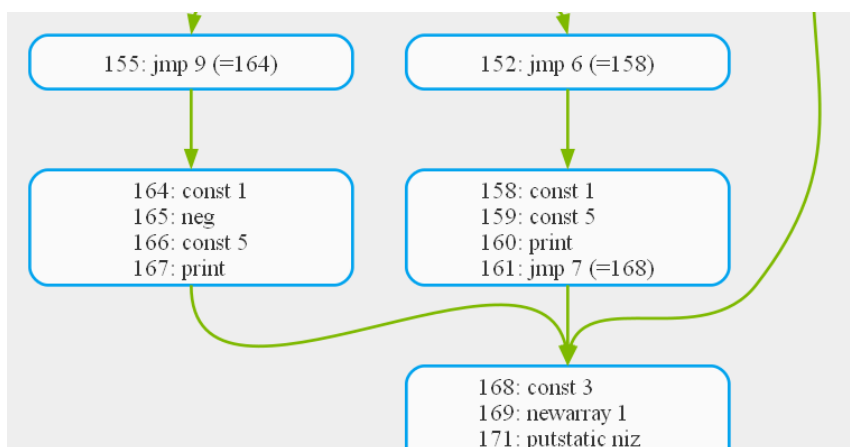
slika 3.2.4

Pozivi virtuelnih funkcija izgledaju kao na slici 3.2.5.



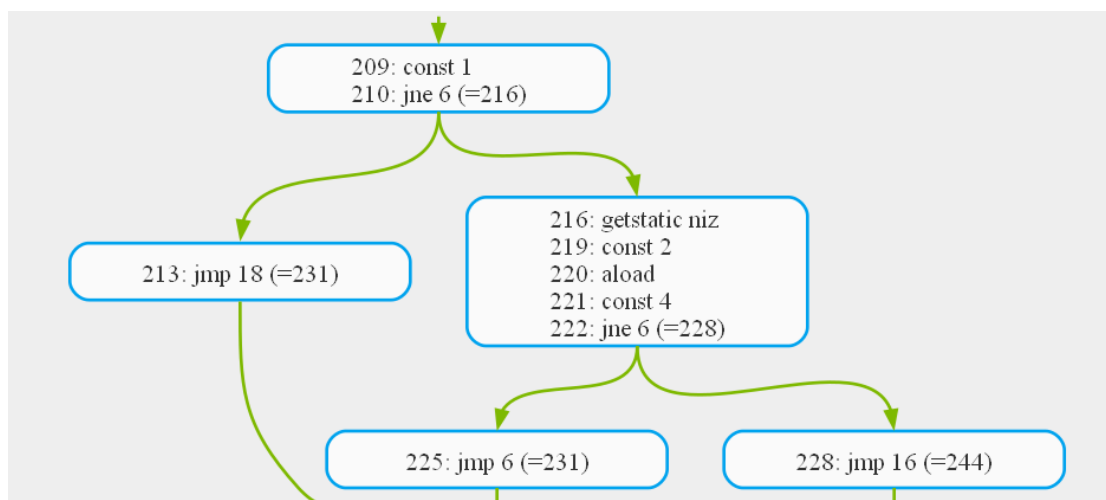
slika 3.2.5

Bezuslovnim skokom se završava osnovni blok i prelazi se na određenu adresu (slika 3.2.6)



slika 3.2.6

Uslovnim skokom se završava osnovni blok, a dalje je moguće ići ili na određenu adresu skoka u slučaju ispunjenog uslova, ili na sledeću instrukciju u slučaju neispunjenog uslova. (slika 3.2.7)



slika 3.2.7

3.3. Korišćenje tabele simbola

U okviru kompajlerske aplikacije (prvenstveno se odnosi na projekte na predmetu Programski prevodioci 1) nakon kompajliranja moguće je pozvati sledeću metodu klase TabDump:

```
public static void saveTab(String fileName);
```

Argument ove funkcije je naziv fajla u koji će biti sačuvano trenutno stanje klase Tab (tabele simbola).

Aplikaciju je moguće pokrenuti sa ili bez izabranog fajla koji sadrži informacije o tabeli simbola. U nastavku slede neke uporedni primeri izgleda grafa sa i bez tabele simbola.

- lokalne promenljive

```
1: enter 2 2  
4: load z (sum)  
5: load d (sum)  
6: add  
7: exit  
8: return
```

```
1: enter 2 2  
4: load 0  
5: load 1  
6: add  
7: exit  
8: return
```

```
177: enter 2 2  
180: load this (Kvadar.postaviTezinu)  
181: load tezina (Kvadar.postaviTezinu)  
182: putfield 1  
185: load this (Kvadar.postaviTezinu)
```

```
177: enter 2 2  
180: load 0  
181: load 1  
182: putfield 1  
185: load 0
```

- globalne promenljive

```
90: putstatic ibool  
93: const 0  
94: putstatic i  
97: getstatic i  
100: const 2
```

```
90: putstatic 3  
93: const 0  
94: putstatic 1  
97: getstatic 1  
100: const 2
```

- statičke metode

```
200: call sum (= 1)  
203: const 1  
204: add  
205: astore  
206: call verify (= 48)
```

```
200: call -199 (=1)  
203: const 1  
204: add  
205: astore  
206: call -158 (=48)
```

4. REALIZACIJA REŠENJA

Sistem je strukturiran u tri ključna segmenta. Prvi segment predstavlja grafički deo aplikacije, razvijen u programskom jeziku Java pomoću Swing biblioteke. Drugi segment fokusira se na efikasnom čuvanju i učitavanju tabele simbola iz tekstualnog fajla. Finalni segment aplikacije posvećen je parsiranju izvornog koda, s posebnim naglaskom na formiranje grafa kontrole toka podelom na instrukcije i osnovne blokove.

4.1. Struktura rešenja

Paket	cfg.ui	cfg.tab	cfg.structure
Klase	<i>SettingsWindow</i> <i>MainWindow</i> <i>HDIcon</i>	<i>TabDump</i> <i>TabVisitor</i> <i>TabLoader</i>	<i>Parser</i> <i>Instruction</i> <i>Block</i>

4.2. Učitavanje parametara

Za realizaciju učitavanja svih potrebnih parametara, koji su objašnjeni u poglavlju 3.1, korišćena je Swing biblioteka. JFileChooser klasa je korišćena za prpetragu fajlova.

4.3. Čuvanje i učitavanje tabele simbola (opciono)

Tabela simbola se čuva pomoću metode `saveTab(String fileName)` unutar `TabDump` klase. Za njenu implementaciju korišćena je klasa `TabVisitor` koja je izvedena iz klase `SymbolTableVisitor` iz `symboltable` biblioteke. Ta klasa obilazi sve čvorove u tabeli simbola `Tab` i pretvara ih u tekstualni zapis.

Format u kojem su sačuvane informacije iz tabele simbola je sledeći:

kind name: type, adr, level

gde je:

- **kind** – vrsta simbola (const, var, type, meth, fld, elem ili prog)
- **name** – naziv simbola
- **type** – tip simbola, tj. povratna vrednost metode
- **adr** – adresa metode u kodu, tj. redni broj deklarisanja promenljiv i polja klasa
- **level** – koliko argumenata metoda ima, odnosno, da li je lokalna ili globalna promenljiva

Za učitavanje tabele simbola, radi se obrnuti postupak. Iz tekstualnog fajla se čita red po red i svaki red parsira i pročitani simbol se ubacuje nazad u tabelu simbola Tab.

Na slici 4.3.1 može se videti početak parsiranja svake linije sa informacijama o tabeli simbola.

```
public static void decode(String filePath) {
    String line;
    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {

        Tab.currentScope = new Scope(null);
        openedScopeObjStack.push(null);

        String currentFunction = "";
        while ((line = br.readLine()) != null) {
            if (isEmpty(line)) continue;

            String regex = "^((\\s*)(\\w+)\\s+([#\\$\\w]+):\\s*(\\w*[\\[?\\]]?),\\s*(-?\\d+),\\s*(-?\\d+)\\s*$";
            Pattern pattern = Pattern.compile(regex);
            Matcher matcher = pattern.matcher(line);

            if (matcher.find()) {
                int tabCount = matcher.group(1).length() / 3;

                for(int i = scopeLevel - tabCount; i > 0; i--) {
                    if (openedScopeObjStack.peek().getKind() == Obj.Type) {
                        currentOwner = null;
                    }
                    if (openedScopeObjStack.peek().getKind() != Obj.Prog) {
                        Tab.chainLocalSymbols(openedScopeObjStack.pop());
                        Tab.closeScope();
                        scopeLevel--;
                    }
                }

                String kindString = matcher.group(2);
                int kind = kindMap.get(kindString);

                String name = matcher.group(3);
                String type = matcher.group(4);
                int adr = Integer.parseInt(matcher.group(5));
                int level = Integer.parseInt(matcher.group(6));
            }
        }
    }
}
```

Slika 4.3.1

4.4. Parsiranje objektnog fajla

Instrukcije imaju sledeću strukturu.

```
public class Instruction {  
    public int adr;  
    public int size;  
    public String instr;  
    public String arg;  
    public String arg2;  
}
```

Potpisi konstruktora:

```
Instruction(int adr, int size, String instr, String arg, String arg2);
```

```
Instruction(int adr, int size, String instr, String arg);
```

```
Instruction(int adr, int size, String instr);
```

Pored toga, u slučaju **load/store**, **getstatic/putstatic**, **enter** i **call** instrukcija, ukoliko je učitana tabela simbola, naziv argumenta se zamenjuje sa nazivom iz tabele simbola.

Osnovni blokovi imaju sledeću strukturu:

```
public class Block {  
    public int startAdr;  
    public int size;  
    public ArrayList<Instruction> instructions;  
}
```

Nakon uspešno parsiranja objektnog fajla sledeće informacije su oformljene:

- **instructions**: `ArrayList<Instruction>` // lista svih instrukcija
- **functions**: `ArrayList<ArrayList<Block>>` // lista funkcija, gde jednu funkciju čini lista blokova (poređanih po adresi)
- **connections**: `HashMap<Integer, Set<Integer>>` // sa neke adrese, na koje se sve skače (odnosi se samo na uslovne i bezuslovne skokove)
- **calledFunctionFromByFunction**: `HashMap<Integer, HashMap<Integer, Integer>` // unutar jedne funkcije, na kojoj adresi se nalazi poziv koje funkcije (početna adresa)
- **invokevirtualByFunction**: `HashMap<Integer, HashMap<Integer, String>>` // unutar jedne funkcije, na kojoj adresi se nalazi poziv neke virtuelne funkcije

4.5. Kreiranje grafičke reprezentacije grafa

Nakon što su se sve prethodne vrednosti oformile, za svaku pojedinačnu funkciju se kreira jedan .dot fajl iz kojeg se kreira grafički prikaz grafa u .png formatu.

Primer izgleda jednog .dot fajla:

```
digraph G {
  graph [fontsize=16, bgcolor="#eeeeee"]
  edge [fontsize=16]
  node [fontsize=16]
  node [width=3 shape=box]
  ranksep = 0.75
  nodesep = .5
  start [nojustify=true label="zero" shape=ellipse style="rounded,filled" fillcolor="#fafafa" fontcolor="#323232"
  color="#feb800" penwidth=2];
  start:s -> 1:n [color="#feb800" penwidth=2 style="dotted"];
  1 [nojustify=true label="1: enter 0 0\|2: const 0\|3: exit\|4: return\|l" style="rounded,filled" fillcolor="#fafafa"
  fontcolor="#323232" color="#00a3ee" penwidth=2];
}
```

4.6. Prikaz kroz interaktivnu aplikaciju

Nakon oformljenih grafičkih prikaza grafova, otvara se novi prozor sa interaktivnom aplikacijom koja je implementirana unutar klase CfgMainWindow.

Za implementaciju CfgMainWindow klase korišćena je Java Swing biblioteka. Radi čuvanja dobre rezolucije grafova u .png formatu implementirana je klasa HDIcon.

5. ZAKLJUČAK

U ovom radu je prikazan razvoj interaktivne aplikacije za generisanje grafa kontrole toka mikroJava bajtkoda. Aplikacija je pisana u programskom jeziku Java. Pored toga, korišćen je DOT Graphviz jezik za generisanje grafičke reprezentacije grafa. Cilj rada je bio napraviti aplikaciju koja će omogućiti pregled osnovnih blokova pojedinačnih funkcija i toka kontrole.

6. LITERATURA

- [1] Graphviz DOT Language <https://graphviz.org/doc/info/lang.html>
- [2] Izvorni kod <https://github.com/ksendzo/MicroJavaCFG>
- [2] Java <https://docs.oracle.com/en/java/>
- [4] Mikrojava virtuelna mašina <http://ir4pp1.etf.rs/Vezbe/MikrojavaVirtuelnaMasina.pdf>
- [5] Symboltable biblioteka <http://ir4pp1.etf.rs/Domaci/symboltable-1-1.jar>
- [6] Udžbenik Programski prevodioci 1, Elektrotehnički fakultet u Beogradu
http://ir4pp1.etf.rs/Predavanja/pp1_udzbenik.pdf