Sistem za generisanje programskih segmenata za ispitivanje u oblasti vremenske složenosti algoritama

Đorđe Pešić, Marko Mišić, Jelica Protić, Milena Vujošević Janičić

***Apstrakt*—Današnje IT obrazovanje je veoma važno, pa se zato puno ulaže u razvoj metoda i alata za pomoć studentima prilikom učenja i nastavnom osoblju prilikom održavanja kurseva. Ovaj rad prikazuje prototip sistema za generisanje programskih segmenata, koji se mogu koristiti za ispitivanje studenata u oblasti vremenske složenosti algoritama. Kako se sistem i dalje usavršava, dat je opis do sada razvijenih delova, kao i pogled na dalje pravce usavršavanja. Uveden je način modeliranja procesa izgradnje programskog segmenta koji koristi šablone i pravila. Programski segment nastaje od skupa šablona koji se kombinuju pomoću određenog broja pravila. Šablon je jednostavan programski segment, dok pravilo definiše način kombinovanja i zavisnosti po podacima, ukoliko postoje. Rad sistema je prikazan na primeru.**

***Ključne reči*—Vremenska složenost; Generisanje koda; XML tehnologije.**

I. Uvod

U poslednjih nekoliko godina, način ispitivanja studenata na masovnim ispitima iz oblasti računarstva na Elektrotehničkom fakultetu Univerziteta u Beogradu se promenio i sve češće dobija formu testova. Usled velikog broja ispitnih rokova propisanih od strane Univerziteta u Beogradu, testove je potrebno relativno često sastavljati. Stoga se teži automatizaciji određenih aspekata sastavljanja testova, kako bi se smanjio napor i pritisak na nastavno osoblje. Postupak je u literaturi poznat pod imenom *automated test assembly*, a podržan je brojnim komercijalnim i istraživačkim softverskim rešenjima, koja koriste metode i algoritme veštačke inteligencije, logičkog programiranja, kao i genetske algoritme [1].

Način sprovođenja testova može biti na papiru, pa se u tom slučaju odgovori unose na posebno dizajnirane obrasce koji se skeniraju, softverski analiziraju i automatski boduju [2]. Drugi način podrazumeva testiranje na računarima, najčešće korišćenjem sistema za elektronsko učenje, kao što je Moodle, koji se pokazao uspešnim na većem broju kurseva iz oblasti računarstva [3]. U nekim oblastima, kao što su programiranje, algoritmi i strukture podataka i sl., moguće je parametrizovati tekst pitanja [3], [4], a određene oblasti programiranja su pogodne i za automatizovano generisanje ispitnih pitanja.

U ovom radu prikazan je sistem za generisanje programskih segmenata iz oblasti analize složenosti algoritama. Generisani segmenti se mogu koristiti za pravljenje ispitnih pitanja, a potencijalno i za podršku učenju koncepata iz ove oblasti. Konceptualni model sistema, kao i motivacija za proces automatizacije generisanja programskih segmenata iz oblasti analize složenosti algoritama su izloženi u [5]. Modeliranje i skladištenje programskih segmenata korišćenjem XML-a je prikazano u [6]. Kako se sistem i dalje usavršava, ovaj rad će prikazati do sada razvijene delove, dok će biti dati pogledi na pravce daljeg usavršavanja.

Rad je podeljen na šest poglavlja. U drugom poglavlju je izložena postavka problema i osnovni teorijski koncepti u vezi sa generisanjem programskih segmenata. Treći deo prikazuje strukturu realizovanog sistema, dok se četvrto poglavlje bavi šablonima i pravilima koji čine osnovu rada sistema i sadrži opis najbitnijih novina koje rad uvodi. Prikaz rada sistema na jednom primeru je dat u petom poglavlju, nakon čega sledi zaključak sa pravcima daljeg istraživanja.

II. Postavka problema

Analiza složenosti algoritama je teorijska oblast koja proučava vremensku ili prostornu složenost algoritama [7]. Vremenska složenost predstavlja modelovanje vremena izvršavanja nekog algoritma u funkciji dimenzije problema *n*, a izražava se pomoću „veliko O“ notacije [7]. Prostorna složenost se odnosi na potrebu za memorijskim prostorom u funkciji dimenzije problema *n*, ali nije tema ovog rada. Da bi pojednostavili određivanje vremenske složenosti i sam rad sistema, uvedena je pretpostavka da će konačni programski segment uvek imati složenost O(f(n)), gde je *n* dimenzija problema, a *f* je funkcija vremenske složenosti po promenljivoj *n*.

Da bismo mogli softverski generisati programske segmente određene vremenske složenosti, prvo je potrebno na neki način apstrahovati njihovu strukturu. Osnovne gradivne elemente programskih segmenata čine petlje i izrazi. Grananja su uvedena kao operacija nad gradivnim elementima i razmatrana je “if” naredba. Pozivi procedura još uvek nisu

Đorđe M. Pešić, Univerzitet u Beogradu, Elektrotehnički fakultet, Bulevar kralja Aleksandra 73, Beograd, Srbija, Institut RT-RK za sisteme bazirane na računarima, Novi Sad, Srbija
(e-mail: djordje.pesic@rt-rk.com)
Marko J. Mišić, Univerzitet u Beogradu, Elektrotehnički fakultet, Bulevar kralja Aleksandra 73, Beograd, Srbija
(telefon: 381-11-3218-392, e-mail: marko.misic@etf.bg.ac.rs)
Jelica Ž. Protić, Univerzitet u Beogradu, Elektrotehnički fakultet, Bulevar kralja Aleksandra 73, Beograd, Srbija
(telefon: 381-11-3218-456, e-mail: jelica.protic@etf.bg.ac.rs)
Milena M. Vujošević Janičić, Univerzitet u Beogradu, Matematički fakultet, Studentski Trg 16, Beograd, Srbija
(e-mail: milena@matf.bg.ac.rs)

razmatrani, ali se njihovo uvođenje planira u daljem razvoju.

U novoj implementaciji sistema, kao unapređenje je prepoznata potreba da se sami izrazi kombinuju među sobom, kao i sa petljama, čime se omogućava da se u telu petlje nađe i izraz koji ne utiče na konačnu vremensku složenost, čime se obogaćuje skup konačnih segmenata koje je moguće generisati. Sami segmenti se mogu ugneždavati, sekvencirati i selektovati. Selekcija se vrši pomoću "if" naredbe. Sva tri tipa kombinovanja mogu se vršiti na dva načina:

1) Kombinovanje dva potpuno nezavisna segmenta, čime se dobija složeni segment, čija je složenost jednaka proizvodu složenosti segmenata koji ga sačinjavaju u slučaju ugnježdavanja ili većoj od dve složenosti u slučaju sekvenciranja i selekcije. Dva segmenta su potpuno nezavisna ako između njih ne postoji zavisnost po podacima.
2) Kombinovanje segmenata sa uspostavljanjem međusobne zavisnosti. Zavisnosti po podacima između segmenata uspostavlja korisnik sistema i one su rezultat korisnikovog intelektualnog rada. Definisane su vezama između portova [6]. Port programskog segmenta je promenljiva čija se vrednost menja unutar tela segmenta, promenljiva koja se nalazi u okviru izraza koji predstavljaju granice petlje ili koja predstavlja iterator petlje.

Sistem treba da bude sposoban da generiše što veći skup različitih programskih segmenata, od baze raspoloživih segmenata korišćenjem operacija kombinovanja. Kako bi se predupredile moguće greške, vremenska složenost dobijenih segmenata treba da bude izračunata od strane sistema, ukoliko je to moguće. Računanje vremenske složenosti programskog segmenta teorijski je nemoguće u opštem slučaju. Ovo je direktna posledica problema zaustavljanja [8].

Problem zaustavljanja je formulisan na sledeći način: za proizvoljan računarski program i njegov ulaz, uz nepostojanje bilo kakvih ograničenja vezanih za resurse i vreme izvršavanja, odrediti da li će se program završiti ili će se izvršavati beskonačno dugo. Tjuring je prvi dokazao da je ovaj problem neodlučiv [8]. Ako je za proizvoljan program nemoguće odrediti da li će se završiti, onda je nemoguće proceniti i broj naredbi koji će se izvršiti, pa je samim tim i nemoguće odrediti njegovu vremensku složenost. Iz ovih razloga je potrebno pretpostaviti određena ograničenja vezana za programske segmente, da bi njihova vremenska složenost mogla biti izračunata. Ta ograničenja su vezana za strukturu segmenta i kompleksnost izraza koji se u njemu javljaju.

Radi inicijalnog implementiranja opisanog kombinovanja segmenata, uveden je koncept strategije generisanja. Strategija generisanja segmenata predstavlja način na koji je segment izgrađen: način kombinovanja, odabir tipova, broj i međusobna interakcija petlji koja generiše ukupan broj iteracija [4]. Model podataka, korišćen u prethodnoj fazi razvoja, koji služi za definisanje strategija [5] ima problem sa fleksibilnošću, jer definisanje strategije uključuje kompletan opis kombinovanja i vezivanja portova u jednom XML fajlu. Bilo koja promena rezultuje novom strategijom, čak i ako se suštinski ne menja složenost rezultujućeg segmenta. Na primer, ukoliko nova strategija ima iste operacije kombinovanja i iste veze između portova, a naredbe koje ne utiču na složenost se razlikuju, onda zapravo imamo istu strategiju. Ako se dve strategije razlikuju samo po imenima promenljivih, onda su te dve strategije jednake.

Sistem treba da omogući da se strategije definišu na generički i fleksibilniji način tako da se sa što manje definisanih strategija može generisati što veći broj programskih segmenata. Rešenje je da osnovne gradivne jedinice segmenta umesto petlji sa njihovim telima budu izrazi i prazne petlje. Ovakav pristup je osmišljen i prikazan uokviru ovog rada i daje veću fleksibilnost u izgradnji segmenta u odnosu na prethodno evaluirane pristupe [5], [6]. Tela petlji će nastati tako što će se sami izrazi definisati kao posebni šabloni i onda će se pomoću pravila vršiti ugneždavanje potrebnog broja izraza unutar praznog kostura petlje. Pravila i šablone kreira korisnik.

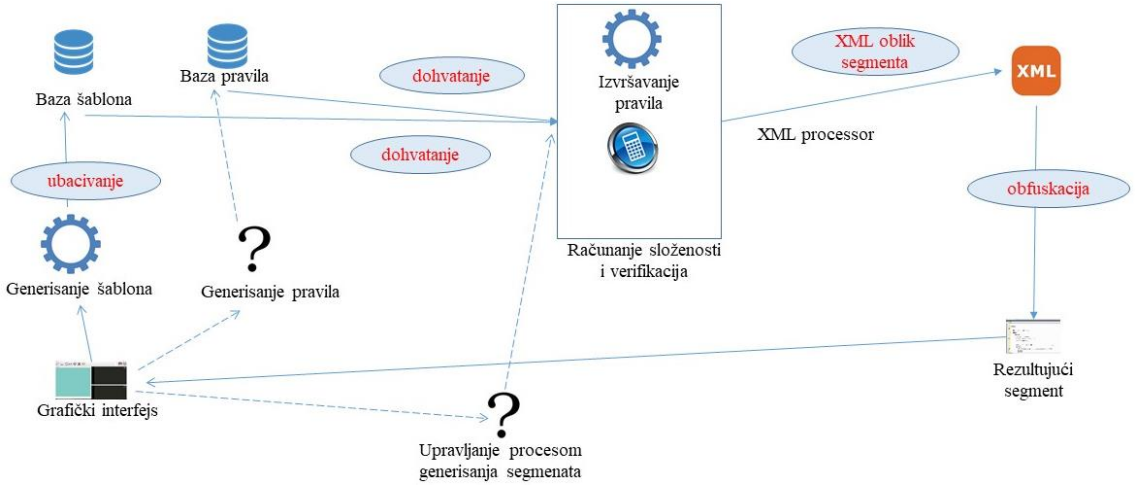
Sledeći korak u razvoju je modeliranje izgradnje segmenta i omogućavanje softverskog računanja vremenske složenosti. A. W. Bierman je u svom radu [9] prikazao jedan način predstavljanja programskog segmenta pomoću niza apstraktnih izraza, koji se dobijaju pomoću skupa pravila primenjenih na konkretan programski kod. Ovaj metod je razvijen i korišćen kako bi se studentima lakše objasnilo određivanje vremenske složenosti, ali su određene ideje iskorišćene i u kontekstu ovog rada.

U ovom radu usvojen je praktično obrnut pristup: polazeći od skupa definisanih pravila i šablona koji će se kombinovati pomoću tih pravila, dobijamo rezultujuće programske segmente. Pored toga, dobijamo i trag o tome kako su ti segmenti sastavljeni, u smislu operacija koje su korišćene i uspostavljenih zavisnosti po podacima. Stoga, poznajući vremenske složenosti šablona, u određenim slučajevima možemo matematički izračunati vremensku složenost dobijenog programskog segmenta.

Vremenska složenost se u trenutnoj fazi razvoja softvera računa kada su segmenti koji se kombinuju međusobno nezavisni po podacima. Tada, u slučaju ugneždavanja, vremenska složenost rezultujućeg segmenta je proizvod vremenskih složenosti segmenata koji se ugneždavaju. U slučaju sekvenciranja i selekcije, rezultujuća složenost je jednaka je maksimumu vremenskih složenosti segmenata koji se kombinuju [7]. Neka su vremenske složenosti tih segmenata O(f(n)) i O(g(n)). Složenost O(f(n)) je veća od složenosti O(g(n)) ako važi

$$\lim_{n->\infty} \frac{O(f(n))}{O(g(n))} = \infty \quad (1)$$

Ostvareni rezultat nove koncepcije sistema, prikazane u ovom radu, je veći skup segmenata koji se mogu generisati uz mogućnost prikazivanja procesa tog generisanja, što se potencijalno može upotrebiti i za učenje oblasti, jer bi studenti analizom sastavljanja segmenata mogli da razumeju i koja je njihova vremenska složenost. Prednost nove koncepcije sistema je ponovna upotrebljivost šablona u većem broju pravila, kao i ponovna upotrebljivost samih pravila. Ovako se drastično povećava skup mogućih generisanih segmenata. Još jedna prednost je mogućnost samodopunjavanja baze šablona, jer svaki segment koji se generiše ima format šablona i može se smestiti u bazu.

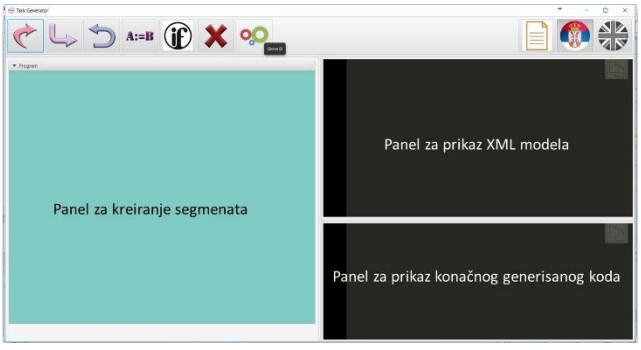


Slika 1. Grafički prikaz modula sistema i veza između njih

III. STRUKTURA SISTEMA

Sistem je izgrađen modularno. Moduli i veze između njih prikazane su na Slici 1. Komponente sistema su: grafički korisnički interfejs, modul za generisanje šablona, modul za generisanje pravila, skup postojećih šablona koji se kombinuju u procesu generisanja, skup pravila čijom obradom se vrši generisanje, modul za izvršavanje pravila, računanje složenosti i verifikaciju, modul za upravljanje procesom generisanja segmenata i modula za procesiranje XML-a. Za model podataka segmenata, koristi se unapređena XSD šema, bazirana na onoj koja se koristila za opisivanje strategija [6]. Za učitavanje svih XML fajlova, kao i za njihovo generisanje koristi se alat JAXB [10].

Grafički interfejs (Slika 2) je jedna od novina u odnosu na ranije implementacije sistema [5], [6] i razvijen je sa ciljem da se olakša upotreba sistema, kao i njegovo testiranje. Njegova trenutna uloga u sistemu je da omogući vizuelno prikladan unos šablona u odgovarajuću bazu. Grafički interfejs je podeljen na traku sa komandama, panel za sastavljanje šablona, panel za prikaz generisanog XML modela i panel za prikaz konačnog generisanog koda. Na Slici 2 su naznačeni delovi grafičkog interfejsa.



Slika 2. Grafički interfejs

Postojeće komande omogućavaju dodavanje petlji, izraza, selekcija, brisanje segmenta, pokretanje generisanja, učitavanje postojećeg šablona, snimanje generisanog koda u datoteku, promenu jezika interfejsa na srpski i na engleski. U gornjim desnim uglovima panela za prikaz generisanog XML-a i generisanog koda nalazi se komanda sa smeštanje sadržaja panela na *clipboard*. Pomoću trenutne verzije grafičkog interfejsa moguće je dopunjavanje baze šablona, dok će dalji razvoj omogućiti i dopunu baze pravila.

Šabloni se definišu tako što se odabere odgovarajući segment u okviru trake za komande i zatim postavi na panel za kreiranje šablona. Postavljanje segmenata u panel se vrši klikom na prostor unutar panela. Rezultat je grafička predstava željenog segmenta. U zavisnosti od mesta postavljanja, segmenti se mogu sekvencirati i ugnežđavati Kada se pojavi grafička predstava segmenta, onda se popunjavaju tekstualna polja sa atributima. Grafičkim elementima se može menjati redosled na panelu. Interfejs je implementiran korišćenjem JavaFX [11] tehnologije.

XML procesor, kompletno razvijen u [5], vrši pretvaranje XML modela u konačni segment. Konačni segment je zapisan u konkretnom programskom jeziku. Tokom procesiranja se vrši i obfuskacija programskog segmenta. Obfuskacija je proces modifikovanja delova programskog segmenta, na način koji ne utiče na konačnu vremensku složenost. Time se na osnovu kreiranih šablona i pravila može dobiti veći broj različitih programskih segmenata iste funkcionalnosti i složenosti. Načini vršenja obfuskacije su sledeći:

1) Izostavljanje granica petlje: u ovom slučaju sistem sam odabere promenljivu koja će biti granica. Ime promenljive se odabira na slučajan način iz skupa dostupnih imena. Kada se skup iscrpi, onda se na imena dodaju i brojevi. Onemogućeno je ponavljanje već upotrebljenih imena.
2) Izostavljanje iteratora u „for“ petlji: u ovom slučaju sistem sam odabere ime za iterator, na slučajan način.
3) Izostavljanje operanda u izrazima: u ovom slučaju sistem sam odabere imena promeljivih koje će biti operandi u izrazu.

Da bi proces obfuskacije rezultovao korektnim programskim segmentom, promenljive koje sistem generiše se ne smeju pojavljivati u portovima, zato što se imena tih promeljivih generišu na slučajan način i korisnik kada definiše portove ne zna unapred njihovo ime i samim tim ne može definisati ispravnu zavisnost po podacima.

Modul za upravljanje procesom generisanja segmenata je deo sistema koji tek treba da se razvije. Razvijanje ovog modula omogućiće kreiranje pravila kroz grafički interfejs. Takođe će biti moguće generisati segmente na nekoliko načina. Jedan način bi bio dirigovana sinteza u kojoj bi

korisnik zadao ciljanu složenost, pa bi ga sistem vodio kroz proces generisanja segmenta željene složenosti. Drugi način bi bio da korisnik zada sistemu da izgeneriše segment proizvoljne složenosti, pa bi sistem sam odabirao koje bi šablone i pravila koristio u toku procesa generisanja.

IV. ŠABLONI I PRAVILA

U ovom poglavlju biće dat opis novog modela šablona i pravila kao i algoritam procesiranja samih pravila. Šabloni predstavljaju XML modele segmenata koji se kombinuju od strane modula za izvršavanje pravila. Šablon može biti proizvoljne složenosti, ali se preporučuje da bude jednostavan kako bi se održala fleksibilnost, jer se jednostavniji šabloni mogu koristiti u većem broju pravila. Svaki šablon se čuva u XML datoteci, a XML datoteke sa šablonima se čuvaju u jednom direktorijumu. Uz XML model šablona, upisana je i njegova vremenska složenost, kao i njegovi ulazni i izlazni portovi. Ovo je potrebno kako bi se omogućilo definisanje zavisnosti po podacima. Vremensku složenost šablona zadaje korisnik i njegova je odgovornost da ona bude ispravna.

Šabloni mogu biti parametrizovani, kako bi se omogućila obfuskacija tokom generisanja. Kod petlji se mogu izostaviti granice i iteratori i onda se oni slučajno odabiraju, dok se kod izraza umesto promenljivih stavlja specijalni znak, čime se XML procesoru daje do znanja da se tokom generisanja koda odabere ime promenljive na slučajan način. Pošto izrazi mogu biti proizvoljnog oblika (dok god odgovaraju gramatici jezika koji se koristi za konačni oblik segmenta), napravljena je ANTLR [12] gramatika i izgenerisan je parser za procesiranje izraza. Gramatika je obogaćena detekcijom specijalnog simbola kojim se označava slučajno odabiranje imena promenljive.

Pravilo služi da opiše koje se operacije koriste za kombinovanje šablona i za definisanje zavisnosti po podacima između šablona. Osim što kombinuju šablone, pravila mogu kombinovati rezultate izvršavanja drugih pravila. Ovim se postiže primena više pravila u izgradnji segmenta, tako što se pravila međusobno ulančavaju. Svako pravilo može kombinovati dva šablona, dva izlaza drugih pravila ili kombinacije šablona i izlaza pravila. Navode se imena fajlova sa šablonima ili pravilima. Zatim se definiše operacija koja se koristi (sekvenciranje, ugnežđavanje ili selekcija). Kod selekcije se navodi i šablon koji predstavlja „if“ segment. Na kraju se navode konekcije između portova. Svaka konekcija opisuje koji izlazni port se povezuje na koji ulazni port segmenta. Pravilo se čuva u okviru svog XML fajla. Način pisanja XML fajlova koji čuvaju pravila, definisan je pomoću XML Schema jezika. Svi XML fajlovi sa pravilima se čuvaju u posebnom direktorijumu. Pravila se ručno unose u sistem.

A. Procesiranje pravila

Procesiranje pravila predstavlja jezgro sistema i njime nastaju rezultujući programski segmenti. Rezultat procesiranja pravila je stablo objekata klasa koje su generisane pomoću JAXB alata. Ovo stablo se može koristiti kao ulaz u neko drugo pravilo ili se prosleđuje delu sistema za konačno generisanje koda programskog segmenta. Izlazi pravila se pored toga čuvaju u XML datotekama. Algoritam procesiranja pravila prikazan je sledećim pseudokodom:

```
ruleProcessing(ruleXmlModel){
  if(firstTemplateName is ruleName){
    rule = loadRule(firstTemplateName)
    firstTemplate = ruleProcessing(rule)
  } else
    firstTemplate = loadTemplate(firstTemplateName)

  if(secondTemplateName is ruleName){
    rule = loadRule(secondTemplateName)
    secondTemplate = ruleProcessing(rule)
  } else
    secondTemplate =
    loadTemplate(secondTemplateName)

  switch(operation){
    case sequence:
      result = sequence(firstTemplate,
      secondTemplate)
    case nesting:
      result =
      nesting(firstTemplate, secondTemplate)
    case selection:
       selectionTemplate =
       loadTemplate(selectionTemplateName)
       result = selection(firstTemplate,
       secondTemplate, selectionTemplate)
  }
  calculateComplexity()
  connectPorts()
  return result
}
```

Procesiranje pravila započinje učitavanjem pravila iz njegovog XML fajla. Nakon učitavanja, proveri se da li je na mestu prvog šablona definisana XML datoteka sa šablonom ili XML datoteka sa pravilom. U prvom slučaju, vrši se učitavanje šablona iz njegove XML datoteke, dok se u drugom slučaju vrši učitavanje specificiranog pravila i zatim se pokreće njegovo izvršavanje. Rezultat izvršavanja se prosledi u tekuće pravilo. Na ovaj način je omogućeno ulančavanje pravila. Na isti način se obradi drugi šablon iz definicije pravila. Prilikom učitavanja šablona, vrši se zamena imena varijabli na način koji je definisan u pravilu. Formalna imena promenljivih koja su data prilikom kreiranja šablona, zamenjuju se konkretnim vrednostima. Na ovaj način je postignuta ponovna upotrebljivost šablona u različitim pravilima.

Nakon učitavanja prvog i drugog šablona sledi izvršavanje pravila. Na osnovu tipa specificirane operacije, vrši se sekvenciranje, ugnežđavanje ili selekcija dva šablona. Operacija se vrši povezivanjem dva objektna stabla tako da rezultujuće stablo takođe ima format koji odgovara XML reprezentaciji šablona. Zatim se vrši računanje vremenske složenosti, ukoliko je to moguće. Vremenska složenost se za slučaj ugnežđavanja, računa tako što se vrši simboličko računanje proizvoda dve funkcije složenosti. Za slučaj sekvenciranja i selekcije, koristi se formula (1). Računanje vremenske složenosti se vrši pomoću matematičkog alata Yacas [14] koji je integrisan u sistem.

Poslednji korak je uspostavljanje zavisnosti po podacima, ukoliko one postoje. Uspostavljanje zavisnosti se vrši povezivanjem portova segmenata. U pravilu je definisano koji port segmenta, koji predstavlja prvi šablon, se vezuje na koji port segmenta, koji predstavlja drugi šablon. Povezivanje se realizuje tako što promenljive koje su ulazni portovi dobiju imena odgovarajućih promenljivih koje su izlazni portovi. Na primer: Neka prvi segment ima izlazni port „a“, a drugi

segment ima ulazni port „b“. U pravilu se definiše da se port „a“ vezuje za port „b“. Povezivanje se realizuje tako što se sva mesta u drugom segmentu gde se pojavljuje promenljiva „b“ zamene sa „a“. Ovakav način preimenovanja zahteva kretanje kroz celo rezultujuće objektno stablo i njegovo menjanje pod određenim uslovima. Da bi ovaj postupak bio olakšan, iskorišćen je XPath [13]. XPath je jezik kojim se zapisuju upiti kojima se pretražuje XML datoteka. U implementaciji sistema iskorišćena je JXPath biblioteka koja omogućava da se XPath primeni i na bilo koja objektna stabla, kao i da se ta stabla menjaju, pošto sam XPath jezik ne podržava izmene. XPath upiti se koriste i u ostatku sistema gde god je potrebno pretraživati ili menjati XML datoteke.

V. PRIMER RADA SISTEMA

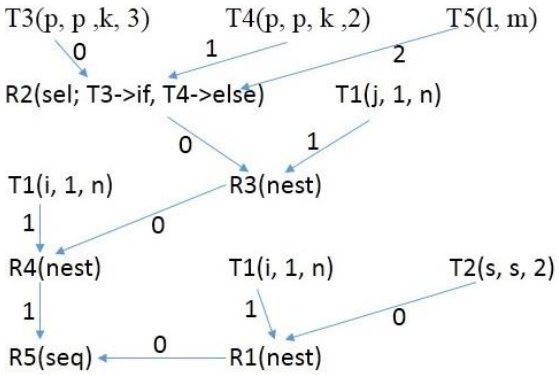
U ovom poglavlju biće dat primer rada sistema. Cilj je demonstrirati dirigovani režim generisanja, gde je potrebno postići kvadratnu složenost pomoću odgovarajućeg povezivanja pet šablona i pet pravila. Skup šablona koji se koriste dat je u Tabeli 1, a korišćena su sledeća pravila:

1) R1(nest, T1, T2): ugnežđavanje šablona T1 i T2.
2) R2(sel, T3->if, T4->else, T5): selekcija šablona T3 i T4 pomoću šablona T5. Šablon T5 predstavlja kostur „if“ naredbe. Šablon T1 se smešta u „if“ granu, a šablon T2 se smešta u „else“ granu.
3) R3(nest, T1, R2): ugnežđavanje šablona T1 i izlaza pravila R2.
4) R4(nest, T1, R3): ugnežđavanje pravila T1 i izlaza pravila R3.
5) R5(seq, R4, R1): sekvenciranje izlaza pravila R4 i izlaza pravila R1.

Navedeni šabloni se povezuju pomoću navedenih pet pravila. Povezana pravila čine stablo i izvršavanjem svih pravila dobijamo rezultujući segment. Pravila mogu biti slična međusobno, i njihova ponovna upotreba i eventualna parametrizacija biće implementirana u modulu za upravljanje procesom generisanja segmenata, u daljem razvoju sistema. Slika 3 pokazuje veze između šablona i pravila. Strelicama je naznačena veza izlaza i ulaza pravila koji su ulančani. Veze formiraju stablo. Izvršavanje pravila kreće od korena stabla i vrši se rekurzivno. Redosled rekurzivnih poziva označen je na slici i vrši se od 0 pa na dalje. Redosled izvršavanja pravila je: R5, R1, R4, R3 i R2. Svaki šablon se instancira sa odgovarajućim promenljivama. Vezivanje aktuelnih i formalnih vrednosti definiše se u pravilu. Aktuelna vrednost je ime promenljive koja se javlja prilikom instanciranja šablona, dok je formalna vrednost ime promenljive prilikom definisanja šablona. Generisanje kreće od pravila R5. U toku njegovog izvršavanja, izvršavaju se pravila R1 i R4. Pravilo R1 ugnezdi šablon T2 u šablon T1 i novi segment se prosledi pravilu R5. Nakon izvršavanja pravila R1 izvrši se pravilo R4, koje ugnezdi izlaz pravila R3 u šablon T1. Izlaz pravila R3 dobija se ugnežđavanjem izlaza R2 u šablon T1. Pravilo R2 koristi šablon T5 kao kostur „if“ naredbe i u „then“ granu smesti šablon T3, a u „else“ granu smesti šablon T4.

TABELA 1: SKUP ŠABLONA KOJI SE KORISTE ZA GENERISANJE ŽELJENOG SEGMENTA

Šablon	Kod
T1(i,a,b)	for i:=a to b do;
T2(a,b,c)	a:=b+c
T3(a,b,c,d)	a:=b+c-d
T4(a,b,c,d)	a:=b-c+d
T5(i,j)	if(i<j) then;



Slika 3. Stablo koje prikazuje veze između šablona i pravila

Programski segment složenosti O(N), koji je rezultat primene pravila R1:

```
for i := 1 to N do
begin
  s := s + 2
end
```

Izlaz pravila R2, složenosti O(1):

```
if l<m then
begin
  p := p + k - 3
end
else
begin
  p := p - k + 2
end
```

Izlaz pravila R3, složenosti O(N):

```
for j := 1 to N do
begin
  if l<m then
  begin
    p := p + k - 3
  end
  else
  begin
    p := p - k + 2
  end
end
```

Izlaz pravila R4, složenosti $O(N^2)$:

```
for i := 1 to N do
begin
  for j := N downto 1 do
  begin
    if l<m then
    begin
      p := p + k - 3
    end
    else
    begin
      p := p - k + 2
    end
  end
end
```

Izlaz pravila R5 koji predstavlja konačni programski segment:

```
for i:= 1 to N do
begin
  s := s + 2
end;
for i0 := N downto 1 do
begin
  for j := N downto 1 do
  begin
    if l<m then
    begin
      p := p + k - 3
    end
    else
    begin
      p := p - k + 2
    end
  end
end
```

Vremenska složenost ovog programskog segmenta je $O(N^2)$

VI. Zaključak

U ovom radu prikazan je deo sistema za generisanje programskih segmenata. Sistem je sačinjen od grafičkog korisničkog interfejsa, modula za generisanje šablona, modula za generisanje pravila, skupa postojećih šablona, skupa postojećih pravila, procesora pravila, generatora segmenata i generatora koda. Procesor pravila računa i vremenske složenosti, kada nema zavisnosti po podacima između šablona. Izlaz sistema predstavljaju programski segmenti koji se mogu koristiti kao ispitna pitanja iz oblasti ispitivanja vremenske složenosti algoritama.

U okviru daljeg istraživanja naglasak će biti na razvijanju modula za upravljanje generisanjem programskih segmenata. Ovaj modul treba da realizuje različite scenarije generisanja koda i upravljanje pravilima (njihovo kreiranje i čuvanje u XML formatu, apstrahovanje u cilju smanjenja količine informacija koja se čuva, uz održavanje fleksibilnosti). Takođe će biti razvijeno određivanje složenosti prilikom izvršavanja pravila, kada postoje zavisnosti po podacima. Potrebno je izvršiti detaljno istraživanje i implementaciju određenih slučajeva, pošto u opštem slučaju ne postoji rešenje za ovaj problem.

Zahvalnica

Ovaj rad je delimično finansiran od strane Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije, projekti broj III44009, OI174021 i TR32047, 2017. Autori izražavaju zahvalnost na finansijskoj podršci.

Takođe, zahvaljujemo se studentu master studija Draganu Čečavcu na učešću u razvoju grafičkog korisničkog interfejsa.

Literatura

[1] A. Bošnjaković, J. Protić, D. Bojić, I. Tartalja, *Automating the Knowledge Assessment Workflow for Large Student Groups: A Development Experience*, International Journal of Engineering Education, Vol. 31, No. 4, pp. 1058-1070, Jul, 2015.

[2] M. Mišić, M. Lazić, J. Protić, *A software tool that helps teachers in handling, processing and understanding the results of massive exams*, 5th Balkan Conference in Informatics, pp. 259-262, ACM New York, NY, USA, Novi Sad, Serbia, Sep, 2012.

[3] Drašković D., Mišić M., Stanisavljević Ž., "Transition from traditional to LMS supported examining: A case study in computer engineering", Computer Applications in Engineering Education, Vol. 24, No. 5, pp. 775-786, September 2016., ISSN: 1061-3773, IF 2015: 0.985, DOI: 10.1002/cae.21750

[4] H. Geerlings, W. J. van der Linden, C. A. W. Glas, *Optimal Test Design With Rule-Based Item Generation*, Applied Psychological Measurement, vol. 37, no. 2, pp. 140-161, 2013.

[5] Đ. Pešić, S. Purić, M. Mišić, J. Protić, *Softversko generisanje pitanja i odgovora iz oblasti analize složenosti algoritama za testove na kursevima programianja*, XXII Skup Trendovi razvoja: Nove tehnologije u nastavi, Zlatibor, 16.-19.02.2016, pp. 53-56.

[6] Đ. Pešić, S. Purić, M. Mišić, J. Protić, *Softversko generisanje programskih segmenata baziranih na strategijama modeliranim pomoću XML-a*, ETRAN 2016, Zlatibor, 13-16.6.2016, str RT5.3.1-6, ISBN: 978-86-7466-618-0

[7] R. Sedgewick, P. Flajolet, *An introduction to the analysis of algorithms*, 2nd edition, Addison-Wesley, 2013.

[8] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London mathematical society 2.1 (1937): 230-265.

[9] A. W. Biermann, *A Simple Methodology for Studying Program Time Complexity*, Computer Science Education, vol. 1, no. 4, pp. 281-292,1990.

[10] E. Ort, B. Mehta, *Java Architecture for XML Binding (JAXB)*, http://www.oracle.com/technetwork/articles/javase/index-140168.html, pristupano 4.2.2016

[11] JavaFx, http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[12] P. Terence, *The definitive ANTLR 4 reference*, Pragmatic Bookshelf, 2013.

[13] B. Anders, S. Boag, D. Chamberlin, M. F.Fernandez, M. Kay, J. Robie, J. Simeoun, *Xml path language (xpath).* World Wide Web Consortium (W3C) (2003), available: http://www.w3pdf.com/W3cSpec/XPath/1/xpath20.pdf

[14] A. Z. Pinkus,S. Winitzki, *Yacas: A do-it-yourself symbolic algebra environment*, Artificial Intelligence, Automated Reasoning, and Symbolic Computation. Springer Berlin Heidelberg, 2002. 332-336.

Abstract

Nowadays IT education is very important, so a lot of effort is put into the development of tools for helping students in learning, or teachers in lecturing. This paper describes a prototype of the program segment assembling system. These segments can be used for examining in the field of algorithmic complexity. The system is under development, so this paper presents developed parts and projections for future work. New program segment assembling model, which uses rules and templates is introduced. The segment is built with the set of the templates, which are connected with rules. The template is a simple program segment. The rule defines combining method and data dependencies if they exist. Finally, an example of program segment assembling by the proposed system is given.

The program segment assembling system for examination in the field of algorithmic time complexity

Đorđe Pešić, Marko Mišić, Jelica Protić, Milena Vujošević Janičić