TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Alexandra Põllumäe

Mudelipõhise testimise õppematerjalide ja koduülesande loomine ainele ,,Automaadid, keeled ja translaatorid”

Bakalaureusetöö (9 EAP)

Juhendaja:
Vesal Vojdani, PhD

Tartu 2022

Mudelipõhise testimise õppematerjalide ja koduülesande loomine ainele „Automaadid, keeled ja translaatorid”

Lühikokkuvõte:

Bakalaureusetöö eesmärk on koostada õppematerjalid mudelipõhisest testimisest kursusele “Automaadid, keeled ja translaatorid” ning leida vastus uurimusküsimusele, kas mudelipõhiselt sobib testida kursuse neid ülesandeid, mille taga peituvad olekumasinad? Töös kasutati mudelipõhise testimise tööriista GraphWalker, mille olemus sarnaneb olekumasinatele.

Võtmesõnad:

Mudelipõhine testimine, GraphWalker, õppematerjal, olekumasinad, automaadid.

CERCS:

P175 Informaatika, süsteemiteooria

Creating model-based testing tutorials and homework for course “Automata, languages and compilers”

Abstract:

The aim of the bachelor's thesis is to compile study materials on model-based testing for the course "Automata, languages and compilers" and to find an answer to the research question, is it suitable to use model based testing on the course finite state machine tasks? The tool of choice is the GraphWalker model-based testing tool, which is similar in nature to state machines.

Keywords:

Model based testing, GraphWalker, educational material, finite state machines, automata.

CERCS:

P175 Informatics, system theory

Sisukord

1. Sissejuhatus

Kursus “Automaadid, keeled ja translaatorid” on kohustuslik aine Tartu Ülikooli Informaatika bakalaureusekraadi õppekavas. Kursusel tutvustatakse lõplike automaate leksilise analüüsi kontekstis ning praktikumides tuuakse ka näited, kuidas programme saab olekumasinate abil selgemini kirja panna. Selles kontekstis saab tutvustada mudelipõhist testimist olekumasinate rakendusel.

Bakalaureusetöö eesmärk on anda üldine ülevaade mudelipõhisest testimisest ning koostada õppematerjalid ja ülesanne eelnimetatud kursusele. Kuna mudelipõhist testimist kasutatakse pigem spetsiifilise tarkvara peal ning see ei ole eriti üldlevinud testimise meetod, siis püstitati ka uurimisküsimus: kas mudelipõhiselt sobib testida kursuse “Automaadi, keeled ja translaatorid” neid ülesandeid, mille taga peituvad olekumasinad?

Töös on kasutatud mudelipõhise testimise tööriista GraphWalker, mis sarnaneb kursusel kasutatavale automaatide joonistamise tööriistale JFLAP. Valminud õppematerjalide abil peaks tudeng saama kasutada valitud töövahendit GraphWalker, et kontrollida oma lahendusi nendele kursuse ülesannetele, mis põhinevad olekumasinatel.

Töö on jaotatud kolmeks osaks järgnevalt. Kolmandas peatükis antakse ülevaade mudelipõhisest testimisest, olekumasinatest ning mudelipõhise testimise tööriistast GraphWalker. Neljas peatükk keskendub kursuse olekumasinate ülesannete sobivuse uurimisele ja protsessile. Viiendas peatükis kirjeldatakse koostatud materjale.

2. Mõisted ja terminid

Redaktor (ingl *editor*) on lugemis- ja kirjutusvõimeline rakendus andmeüksuste loomiseks ja muutmiseks [1].

JSON ehk JavaScripti objektide notatsioon (ingl *JavaScript Object Notation*) on lihtne andmevahetusvorming, mis põhineb JavaScripti alamhulgal [2].

HTML ehk hüpertekst-märgistuskeel (ingl *HyperText Markup Language*) on enimlevinud kodeerimissüsteem (tekstivorming) veebidokumentide loomiseks. HTML koodid ehk märgendid määravad ära selle, kuidas veebileht arvutiekraanil välja näeb [3].

Skript (ingl *script*) on käsujada, mida täidetakse ilma kasutajapoolse vahelesegamiseta [3].

Plugin ehk pistikprogramm (ingl *plugin*) on lisandprogramm, mis on hõlpsalt installitav ja laiendab senise programmi võimalusi [4].

Liides (ingl *interface*) on süntaksi mõttes klassile sarnane struktuur, mis kirjeldab komplekti meetodeid [5].

Märgend (ingl *tag*) on vorminduskäsk, mida kasutatakse kõigis teksti dokumentides dokumentide endi või nende osade märgistamiseks [3].

Echo ehk kajakäsk (ingl *echo*) on tekstistringi väljastuse käsk operatsioonisüsteemide kestades ja skriptides [6].

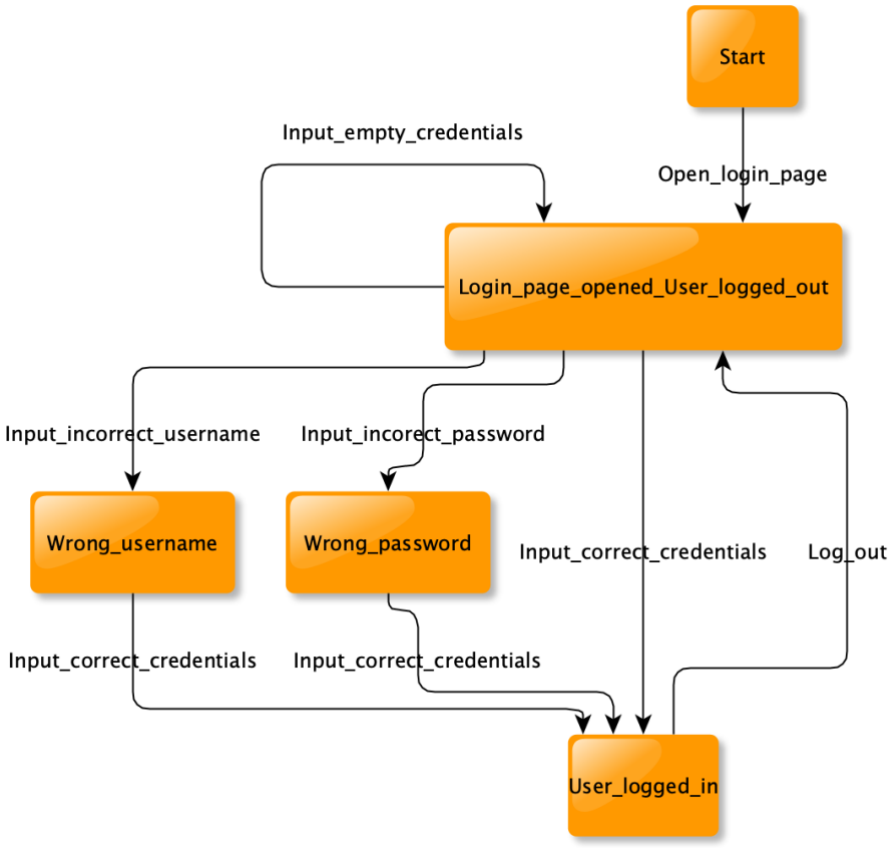
AKT - kursus “Automaadid, keeled ja translaatorid”.

3. Teoreetiline ülevaade

Peatükk annab ülevaate mudelipõhisest testimisest, olekumasinatest ning mudelipõhise testimise tööriistast GraphWalker.

3.1 Mudelipõhine testimine

Tarkvara testimise eesmärk on hinnata toote kvaliteeti ja parandada seda vastavalt leitud puudustele ja probleemidele [7]. Meetodeid tarkvara testimiseks on mitmeid ning mudelipõhine testimine on üks neist. Mudelipõhise testimisega on võimalik testitavat süsteemi hinnata väiksematest üksustest kuni terve süsteemi funktsionaalsuseni välja [7]. Mudelipõhine testimine toimub programmi käitumist kirjeldaval mudelil, kus luuakse automaatseid testjuhtumeid [8]. Kui enamasti kirjutatakse programmi käitumistestimise puhul teste käsitsi nõuete dokumentatsiooni põhjal, siis mudelipõhise testimise puhul luuakse mudel süsteemi oodatud käitumisest [7].



Joonis 1. Mudel, mis iseloomustab sisselogimise funktsionaalsust [9]

Mudel on visuaalne kirjeldus süsteemist, mis näitab selle omadusi väiksemal kujul. Mudel peaks olema võimalikult lihtne, aga piisavalt täpne [7]. See tähendab, et testitava käitumise kirjeldamine ei tohiks võtta liiga palju aega ning peab olema nõuete suhtes lihtne kinnitada. Joonisel 1 on näidatud, milline võib üks mudel välja näha, kui tegemist on süsteemil sisselogimise funktsionaalsusega. Nimelt mudeli järgi on kasutajal võimalik süsteemi sisestada sisse pääsemiseks vajalikud audentimisandmed. Kui sisestatud parool või kasutajanimi pole korrektsed, siis tuleb vastav veateade. Õigete sisendite puhul logitakse kasutaja süsteemi sisse, kus tal on veel omakorda võimalus välja logida ning jõuda taas algolekusse.

Süsteemi kirjeldava mudeli valmimisel genereeritakse sellest mudelipõhise testimise tööriistaga abstraktsete testjuhtumite komplekt [7], mis vastab mudeli tegevustele ja väärtustele. Seejärel teisendatakse testjuhtumite komplektist süsteemi testimiseks tarvilikud test skriptid.

3.2 Mudelipõhise testimise kasulikkus

Kuigi testimisviise on erinevaid, ei ole olemas üht ainuõiget testimise meetodit, mis leiaks kõikidest programmidest kõikvõimalikud vead. Enamasti kasutatakse ühel süsteemil mitut erinevat kvaliteedikontrolli viisi. Kui mudelipõhist testimist rakendada sobivale tarkvarale on tulemuseks testkulude vähenemine, parem koodi katvus testidega ja suurem vigade arvu leidmine, kui võrrelda seda käsitsi testimisega [10]. Testimise ajakulu on võimalik vähendada nii mudelit kui ka genereeritud teste jooksvalt uuendades. Samuti on testijale kasulik testjuhtumite automaatne genereerimine valitud tööriista abil, kuna nende manuaalselt kirjutamine võtab rohkem aega. Lisaks annab mudelipõhine testimine hea visuaalse ülevaate programmi toimimisest.

3.3 Olekumasinad

Aine “Automaadid, keeled ja translaatorid” eesmärk on tudengitele õpetada sügavamat arusaama arvutiprogrammide tähendusest [11]. Üks teemadest selle saavutamiseks on olekumasinad. Lõplik olekumasin on lõplik automaat ja kujutab endast käitumismudelit, mis koosneb olekutest ja üleminekutest ehk siiretest ja toimingutest [3]. Vallaste e-Teatmik märgib veel: “Olek on salvestatud informatsioon mineviku kohta, st sisendite muutuste kohta süsteemi käivitamisest kuni käesoleva hetkeni. Üleminek näitab oleku muutust ja seda

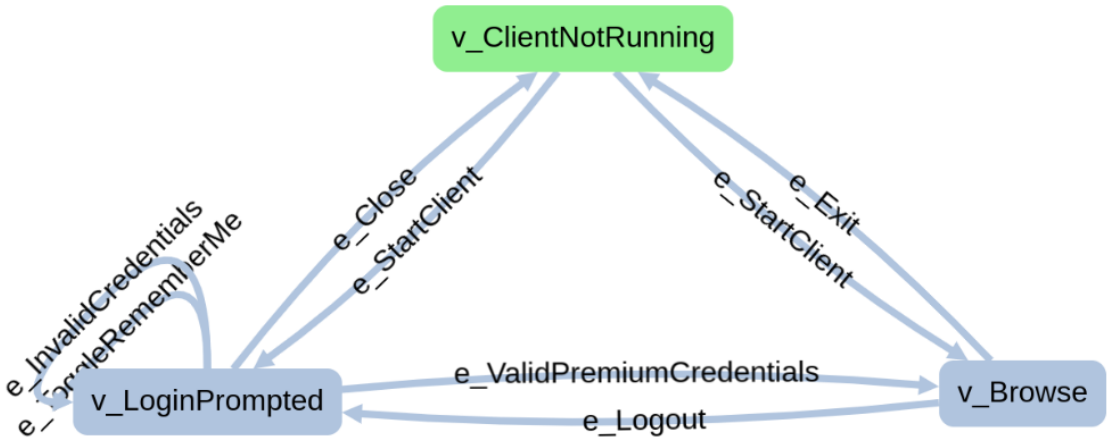
kirjeldatakse tingimusega, mis peab olema täidetud, et üleminek oleks võimalik. Toiming on selle tegevuse kirjeldus, mida antud momendil on vaja teostada" [3].

Lõplike olekumasinaid liigitatakse sõltuvalt sellest, mis toimingud nad võimaldavad. Kursuse põhiteema on lõplikud automaadid, mille ainus toiming on kõige lõpus vastata, kas sisend sobib või mitte. Olekumasinad, mis töö käigus iga sisendtähe korral reageerivad toiminguga, nimetatakse lõplikeks muunduriteks. Selliseid on omakorda kahte liiki: Mealy ja Moore'i masinad. Mealy masina eripära on see, et tema väljundväärtused ehk tegevused sõltuvad nii masina hetkeolekust kui ka sisendite väärtustest [12]. Seevastu Moore'i masina väljundväärtused määratakse ainult hetkeoleku järgi. Kursusel olevate ülesannete puhul on tegemist Mealy masinatega, kuid nende kohandamisel mudelipõhiseks testimiseks GraphWalkeri peal käituvad nad kui Moore'i masinad.

3.4 GraphWalker

Mudelipõhise testimise tööriistu on mitmeid, kuid bakalaureusetöö raames on valitud tööriistaks GraphWalker. GraphWalker on avatud lähtekoodiga ning kirjutatud programmeerimiskeeles Java. Tööriistal on veebipõhine redaktor GraphWalker Studio, kus saab luua ja redigeerida mudeleid ning visuaalselt kontrollida nende õigsust ja eeldatavat toimimist [13]. Kursusel kasutatakse programmi JFLAP, millega saab joonistada automaate. JFLAPi automaatide joonistamise olemus on sarnane GraphWalker Studioga, mistõttu sai just see testimise vahend valituks.

GraphWalkeri toimimine sarnaneb olekumasinatega. Tööriist loeb suunatud graafide kujul mudeleid ja loob neist testteed [14], mida saab ära kasutada, et vastavat programmi testida. Suunatud graaf koosneb tippudest ja suundadega kaartest. Mudelil kujutatud tipud tähistavad olekut ehk kontrolli ning kaared tegevust. Tippudele ja kaartele saab mudelis määrata nimed, millele vastavad hiljem koodis Java meetodid. Nimetused määratakse vastavalt oleku kontrollile või kaare tegevusele. Samuti tehakse nimedega tipud ja kaared eristatavaks, kus tippude (ingl *vertex*) ette pannakse tähemärgid "*v*_" ning kaarte (ingl *edge*) ette "*e*_" (vt Joonis 2).



Joonis 2. Näide GraphWalkeri mudelist sisselogimise toimimisest [13]

Mudelile peab ka defineerima tee generaatori ja lõpetamise tingimuse. Generaator tähistab seda, kuidas peaks GraphWalker mudelit läbima kuni lõpetamise tingimus on täidetud. Võimalusi nendeks on tööriistal mitmeid, kuid vaikimisi on selleks "*random(edge_coverage(100))*". See tähendab, et mudelit läbitakse valides juhuslikke kaari ning testimine lõpetatakse, kui kõik kaared on läbitud. GraphWalker Studios on võimalik visuaalselt mudeli läbimist läbi mängida ning seeläbi selle mudeli korrektsust kontrollida.

Suuremate süsteemide puhul on mõttekas ühe suure mudeli asemel need väiksemateks osadeks jaotada. GraphWalkeris on võimalik erinevad mudelid ühendada jagatud olekuga, märkides ära jagatud nime tingimuse ja nimetades need samasugusteks. Tööriista veel mitmed muud võimalused, mida selles töös ei käsitleta on ära defineeritud GraphWalkeri Wikis[1].

GraphWalkerit on võimalik integreerida testitava süsteemiga mitmel viisil, kuid selles töös käsitletakse tööriista ühendamist Maven projektidega. Selle saavutamiseks tuleb lisada GraphWalkeri sõltuvus POM faili, et ta käituks kui pistikprogrammina kasutatud projektis. GraphWalker Studios valminud mudel või mudelid salvestatakse JSON formaadis ühe failina. Et tööriist saaks loodud mudelist koostada liidese, tuleb see fail salvestada Maven

[1] https://github.com/GraphWalker/graphwalker-project/wiki

projekti ressursi kausta “src/main/resources”. Liides tuleb vastavas testklassis realiseerida ning genereeritud testmeetodidega käibki süsteemi testimine.

4. Kursuse ülesannetele sobivuse uurimine

Järgnev peatükk kirjeldab tööprotsessi, kuidas katsetati mudelipõhise testimise sobivust kursuse ülesannete peal ja analüüsi leitud tulemustest.

4.1 Näide kursuse ülesandest "*HtmlStrip*"

Enamasti rakendatakse mudelipõhist testimist reaalsetele süsteemidele või tarkvaradele. Aine "Automaadid, keeled ja translaatorid" ülesanded, mis on seotud selle tööga, on aga lihtsad koodijupid olekumasinatest. Esimene ülesanne olekutega programmeerimisest on "*HtmlStrip*", mille eesmärk on eemaldada märgendid HTML dokumendist [15]. Nimetatud ülesande puhul katsetas autor kõigepealt, kas on üldse võimalik mudelipõhiselt testida olekumasinaid.

Joonis 3 pealt on näha Java programmeerimiskeeles kirjutatud olekumasin "*HtmlStripMachine*". Sellel masinal on kolm olekut: algolek (*INI*), märgendi olek (*TAG*) ning jutumärkide olek (*QTE*). Masina olekuks (*state*) on esialgu määratatud algolek. Meetod "*process*" (töötlemine) omistab masinale uue oleku vastavalt sisendile ehk toimingule.

```
public class HtmlStripMachine {
   enum State {INI, TAG, QTE }
   private State state = INI;
   public String process(char c) { ... }
}
```

Joonis 3. Koodiplokk "*HtmlStrip*" liidesest [15].

Programmile antakse ette mingi kindel sõne, kuid masina enda töö toimib tähthaaval. Joonisel 4 on kood meetodist "*cleanUp*" (puhasta), kus kutsutakse välja "*process*" meetodit iga sisendi elemendi juures ning tagastatakse töödeldud sõne.

```
   private static String cleanUp(String s) {
       StringBuilder sb = new StringBuilder();
       HtmlStripMachine machine = new HtmlStripMachine();
       for (char c : s.toCharArray()) sb.append(machine.process(c));
       return sb.toString();
   }
```

Joonis 4. Koodiplokk "*HtmlStrip*" masina kasutamisest [15].

4.2 AKT Mealy masinate testimine

Kursusel testitakse olekumasinaid käsitsi nii, et mõeldakse välja kindel sisend kas tava- või erijuhtumitega (vt Joonis 5).

```
Assert.assertEquals("f>5", cleanUp("<a href='>'>f>5</a>"));
```

Joonis 5. Käsitsi kirjutatud test kood.

Mudelipõhiselt saab testida "*process*" meetodi käitumist. AKT Mealy masinate puhul tagastatakse selle meetodi poolt ainult väljund ning olekute sisu ei ole testijale nähtav. Seisund, mida on vaja kontrollida, ongi töödeldud sõne, mistõttu võib servas toimuva tegevuse meelde jätta. Selle jaoks saab ehitada jooksvalt masina poolt töödeldud sõne ning mudeli järgi konstrueerida tulemuse (vt joonis 6). Muutujad nimetatakse vastavalt hea tava järgi "*actual*" (tegelik tulemus) ja "*expected*" (oodatud tulemus).

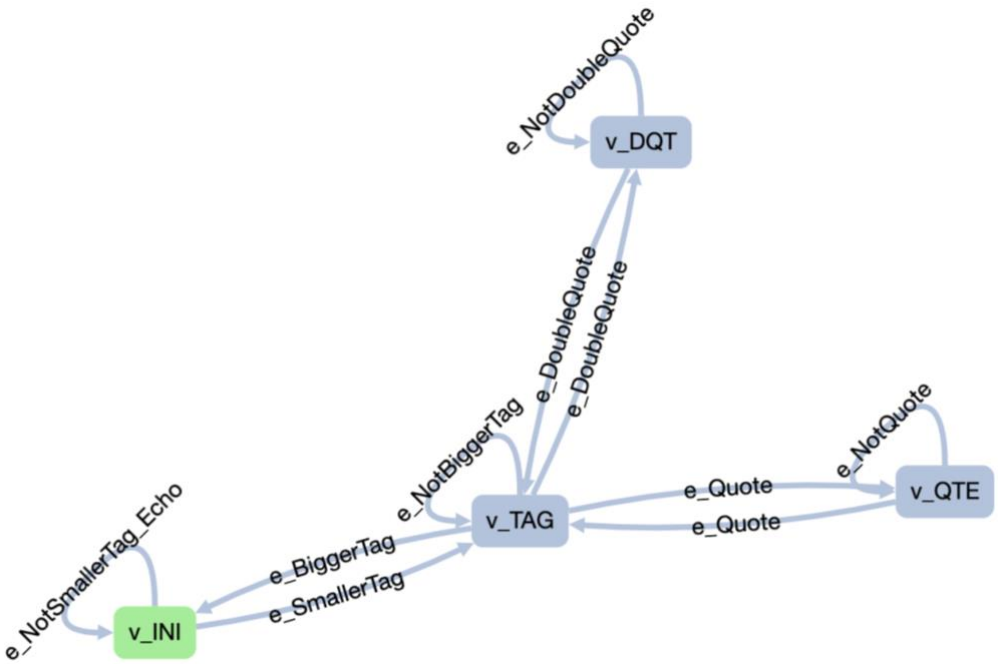
```
@GraphWalker()
public class HtmlStripMBTest extends ... {
 StringBuilder actual = new StringBuilder();
 StringBuilder expected = new StringBuilder();
 HtmlStripMachine machine = new HtmlStripMachine(); …}
```

Joonis 6. Koodi osa mudelipõhise testimise isenditest

Järgmisena tuleb iga serva juures tegevusena mõlemad sõned, "*actual*" ja "*expected*", täiendada vastavalt implementatsiooni ja mudeli põhjal. Joonisel 7 asuv meetod "*processChar*" (tähemärgitöötlus) saab parameetri "*echo*" (kajakäsu) järgi teada, kas antud tähemärk tuleb töödeldud sõnele juurde lisada ning vastavalt sellele täiendab mõlemad sõned. Parameetri väärtus tuleb mudelist (vt Joonis 8), mille põhjal luuakse meetodid servade jaoks. Joonisel 7 on kaks sellist meetodit, mis kutsuvad töötlemismeetodit välja: ühe serva juures tuleb antud tähemärk sõnele juurde lisada, kuid teise serva juhul mitte.

```
private void processChar(char c, boolean echo){
 actual.append(machine.process(c));
 if (echo) expected.append(c);
}
@Override
public void e_NotSmallerTag_Echo() {
 processChar('a', true);
}
@Override
public void e_SmallerTag() {
 processChar('<', false);
}
```

Joonis 7. Abimeetod tähemärgi töötlemiseks ja selle väljakutsumised



Joonis 8. Mudel ülesandest "*HtmlStrip*".

Kui servade juures toimuvad tegevused, siis tippudes peab toimuma kontroll. Selle jaoks on Joonisel 9 olev meetod "*check*" (kontroll), mis võrdleb vastavalt sõnesid.

```
private void check(){
 assertEquals(expected.toString(), actual.toString());
}
```

Joonis 9. Abimeetod sõnede võrdlemiseks

Kuna automaadi olekuks on ainult senimaani töödeldud tekst ehk väljund, siis tegelikult on sõne genereerimise osa seal põhiline. Seega võib töötluse tulemuse kogumise asemel hoopis ehitada ainult sisendi ning igas olekus või kõige lõpus lihtsalt kontrollida meetodit "*cleanUp*" (vt Joonis 10).

```
StringBuilder input = new StringBuilder();
StringBuilder expected = new StringBuilder();

private void processChar(char c, boolean echo){
 input.append(c);
 if (echo) expected.append(c);
}

private void check(){
  assertEquals(expected.toString(), cleanUp(input.toString()));
}
```

Joonis 10. Alternatiivne meetod sõne töötlemiseks

4.3 Lahenduse analüüs

Püstitatud uurimisküsimuse eesmärk oli uurida, kas mudelipõhise testimisega saab testida kursuse "Automaadid, keeled ja translaatorid" teksti töötlemise ülesannete korrektsust. Selle jaoks katsetati läbida kahte näidet. Lisaks ülesandele "*HtmlStrip*" on tudengitele iseseisvaks lahendamiseks ülesanne, kus tuleb liitmise ja lahutamise avaldisi kokku arvutada, näiteks sisendsõne "-5+--8+7+-+-9", mille puhul on tulemuseks 19. Ülesandel on toodud ka üks poolik lahendus.

```
public static int eval(String str) {
   int sum = 0;
   int current = 0;
   int sign = 1;
   for (char c : str.toCharArray()) {
       if (c == '+' || c == '-') {
           if (current != 0) {
               sum += sign * current;
               current = 0;
               sign = 1;
           }
           if (c == '-') sign *= -1;
       } else if (Character.isDigit(c)) {
           current = 10 * current + Character.getNumericValue(c);
       }
   }
   return sum + sign * current;
}
```

Joonis 11. Poolik lahendus avaldise ülesandest.

Joonisel 11 olev kood tagastab vale tulemuse, kui etteantud sisendiks on näiteks "5-0-2". Õige vastus on "3", aga koodis olev lahendus tagastab väärtuse "7". Kui seda testida lihtsa mudeli järgi, kus arvudeks võetakse juhuslik arv 0 ja 100 vahel, siis ainuüksi kõikide servade läbimise lõpetamise tingimusega leitakse viga üsna harva. Kui lõpetamise tingimuseks on läbida 5000 serva, siis avastatakse enamasti viga ja õige koodi testimine võtab 15 sekundit. Õigem on aga luua testimise mudelis eraldi serv oluliste sisendite jaoks. Näiteks kui selle koodi mudelil luua eraldi serv nulli jaoks, siis leitakse viga tihedamini ka lihtsama lõpetamise tingimusega ning 5000 serva läbimise korral õnnestub see igal katsel.

Kirjeldatud piiratud kogemuse põhjal võib väita, et mudelipõhist testimist saab kasutada AKT Mealy masina ülesannete jaoks, aga testide kvaliteet sõltub siiski testimise mudelist ja lõpetamise tingimustest. Kui võtta lihtsalt ülesande spetsifikatsioonis olev olekumasin, siis ei piisa kõikide servade ühekordsest läbimisest, et vigases implementatsioonis viga tuvastada. Seega peab õige testimise mudel sisendit juhtima erijuhtudele, mida ei ole alati võimalik ette teada. Lõpetamise tingimuseks võib kasutada seda, et läbitaks võimalikult

palju servi. See suurendab märgatavalt vea leidmise tõenäosust, aga kui mudeli läbimise tingimusest ei piisa vigade tuvastamiseks, siis see ei anna paremat garantiid, kui lihtsalt juhuslikult teste kirjutades.

5. Loodud materjalid

Bakalaureusetöö raames koostas autor õppematerjalid ja ülesande mudelipõhisest testimisest GraphWalkeril kursusele "Automaadid, keeled ja translaatorid". Nende eesmärk on arendada tudengite silmaringi ning tutvustada uut moodust automaatide testimiseks. Õppematerjalides tutvustatakse tööriista GraphWalker ning tuuakse kaks näidet, kuidas mudelipõhine testimine nimetatud tööriistal toimib. Materjali läbides saab tudeng aru mudelipõhise testimise põhimõttest, oskab kasutada tööriista GraphWalker ning testida sellega kursuse olekumasinatega ülesandeid. Töö autor valmistas kaks repositooriumit: esimene õppejõududele koos lahendustega ning teine tudengitele.

5.1 Ülevaade loodud materjalidest

Kokkupandud õppematerjalid ja ülesanne asuvad kursuse Wiki[2] lehel kolmanda nädala teemade all. Enne ülesande juurde asumist on tudengitel vaja alla laadida ette antud projekti repositoorium GitHubist ning GraphWalker Studio java programm. Kuna selle teema juurde jõudes on kursus juba kolm nädalat kestnud, siis eeldatavasti on tudengitel olemas kõik vajalikud tarkvarad ja nende õiged versioonid ning rohkem seadistusi tarvis teha ei ole.

Esimese õppematerjali näite puhul tahtis töö autor tutvustada mudelipõhist testimist millegi elulise, kuid samas lihtsa näite näol. Leiti, et selle jaoks sobiks koostada õppematerjalid sisselogimise funktsionaalsuse kohta. Seda on lihtne realiseerida ning on piisava keerukusega arusaamiseks. Selle jaoks kirjutas autor lihtsa koodi, et simuleerida sisselogimise funktsionaalsust. Tudengid, kes ülesannet lahendavad, peavad kõigepealt tutvuma etteantud koodiga, et aru saada kuidas süsteem peaks toimima. Seejärel näidatakse, kuidas käivitada GraphWalker Studiot ning kuidas seal luua vajalik mudel olemasolevatest nõuetest. Järgmisena tõstetakse salvestatud mudeli kirjeldus projekti ning sealt luuakse GraphWalkeri pluginaga liidese ehk testteed. Peale seda realiseeritakse liidesega automaatselt genereeritud testteed ning kirjutatakse nendesse testid. Juhendis on lähemalt kirjeldatud ühe testmeetodi realiseerimist ning kuidas teostada süsteemi väljundite kontrolli. Selle põhjal on tudengitel võimalus ülejäänud meetodid ise lahendada ning neid võrrelda

[2] https://courses.cs.ut.ee/2022/AKT/spring

ette antud lahendusega. Viimasena tehakse läbi näide, kus tekitatakse algkoodi viga, et demonstreerida, kuidas mudelipõhise testimisega on võimalik leida üles koodis vigu.

Kui esimene õppematerjal oli selline, mille jaoks tavaliselt mudelipõhist testimist rakendatakse, siis teine näide toimub olekumasinate vaatest. Aluseks on võetud varem mainitud ülesanne “HtmlStrip”. Ka see näide tehakse koos läbi analoogiliselt esimesele ülesandele.

Viimasena tuleb tudengitel iseseisvalt lahendada veel üks kursusel olev ülesanne väärtustamise olekumasinal. Ette on antud ülesande kirjeldus, vigane kood ning poolik mudel. Tudengid peavad ülesande lahendamiseks GraphWalker Studios olemasolevat mudelit täiendama, realiseerima genereeritud testteed, kirjutama nendesse testid ja leidma testimisega üles vea koodis ning selle parandama.

5.2 Edasised arendused

Kuna olekumasinate teema on kursusel “Automaadid, keeled ja translaatorid” võrdlemisi varakult ja materjalid valmisid mitu nädalat hiljem, siis ei olnud võimalik ülesannet tudengite peal läbi viia. Järgmisel õppeaastal on plaanis esitada valminud ülesanne tudengitele lisaülesandena ning vastavalt tagasisidele vajadusel materjale parandada.

Veel saaksid kursuse õppejõud mudelipõhise testimise tööriista GraphWalker kasutada testimise genereerimiseks, kui kontrollid salvestada faili. Sellisel juhul ei ole vajadust kogu aeg tööriista kaudu teste käivitada.

Kokkuvõte

Bakalaureusetöö eesmärk oli luua õppematerjalid mudelipõhisest testimisest kursusele "Automaadid, keeled ja translaatorid". Samuti püstitati uurimisküsimus, kas mudelipõhiselt sobib testida kursuse "Automaadi, keeled ja translaatorid" neid ülesandeid, mille taga peituvad olekumasinad?

Töö annab ülevaate mudelipõhisest testimisest, olekumasinatest ning mudelipõhise testimise vahendist GraphWalker. Uurimisprotsessi käigus tuli välja vastus uurimisküsimusele, et mudelipõhist testimist saab kasutada AKT Mealy masina ülesannete jaoks, kuid testide kvaliteet sõltub testimise mudelist ja lõpetamise tingimustest.

Bakalaureusetöö käigus valmisid õppematerjalid kahe ülesande näitel: üks elulisem näide sisselogimise funktsionaalsusest ning teine AKT olekumasina ülesandest "*HtmlStrip*". Lisaks loodi tudengitele lahendamiseks ka iseseisev ülesanne olekumasina peal, mis tegeleb avaldiste väärtustamisega. Materjali läbides peaks tudeng oskama kasutada tööriista GraphWalker ning testida sellega kursuse olekumasinatega ülesandeid.

Kirjanduse loetelu

[1] AKIT, Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/term/4573-editor-2 (10.05.2022).

[2] AKIT, Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/term/10850-json (10.05.2022).

[3] e-Teatmik: IT ja sidetehnika seletav sõnaraamat. http://www.vallaste.ee/ (10.05.2022).

[4] AKIT, Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/term/2437-plugin (10.05.2022).

[5] Objektorienteeritud programmeerimine - Kursused, Arvutiteaduse instituut. https://courses.cs.ut.ee/2022/OOP/spring/Main/Practice5 (08.05.2022).

[6] AKIT, Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/term/9196-kajakask-echo-kask (10.05.2022).

[7] M. Utting and B. Legeard, "Practical model-based testing: A tools approach," *Elsevier*, 2010, lk 3–49.

[8] J. Jacky, M. Veanes, C. Campbell, and W. Schulte, "Model-Based Software Testing and Analysis with C#," *Cambridge University Press*, 2007, lk 7.

[9] E. Kazakov. "What is Model-based testing - Eugene Kazakov," *Medium*, Dec. 19, 2019. https://medium.com/@yugene1986/what-is-model-based-testing-db3ebde10683 (09.01.2022).

[10] "Model Based Testing Tutorial: What is, Tools & Example," *Guru99*, May 24, 2020. https://www.guru99.com/model-based-testing-tutorial.html#6 (10.01.2022).

[11] Automaadid, keeled ja translaatorid - Kursused, Arvutiteaduse instituut. https://courses.cs.ut.ee/2021/AKT/spring/Main/Syllabus (10.01.2022).

[12] "Unstop," Competitions, Quizzes, Hackathons, Scholarships and Internships for Students and Corporates. https://unstop.com/blog/difference-between-mealy-and-moore-machine (10.05.2022).

[13] GraphWalker. https://graphwalker.github.io/ (10.01.2022).

[14] GraphWalker, *GitHub*. https://github.com/GraphWalker/graphwalker-project/wiki (10.01.2022).

[15] Automaadid, keeled ja translaatorid - Kursused," *Arvutiteaduse instituut*. https://courses.cs.ut.ee/t/akt/Main/Olekumasinad (10.05.2022)

Lisad

I. Materjalide asukohad

Õppematerjalid asjuvad kursuse Wiki lehel: https://courses.cs.ut.ee/2022/AKT/Main/MBT
Repositoorium õppejõududele: https://github.com/alexandrapollumae/AKT-MBT
Repositoorium tudengitele: https://github.com/alexandrapollumae/Model-Based-Testing

II. Litsents

Mina, **Alexandra Põllumäe**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Mudelipõhise testimise õppematerjalide ja koduülesande loomine ainele „Automaadid, keeled ja translaatorid",

mille juhendaja on Vesal Vojdani,

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Alexandra Põllumäe

10.05.2022