TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gedi Georg 213667IABB

Tekstide genereerimine regulaaravaldiste põhjal

Bakalaureusetöö

Juhendaja: Bahdan Yanovich
BSc
Ian Erik Varatalu
MSc

Tallinn 2024

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Gedi Georg

20.05.2024

Annotatsioon

See lõputöö uurib meetodeid tekstide genereerimiseks regulaaravaldiste põhjal, mida on vaja regulaaravaldise mootorite testimisel, keskendudes Microsoft Automata teegi ja selle Rex alamprojekti täiustamisele. Töö eesmärk on luua lahendus tekstide loomiseks, võttes arvesse mitmeid piiranguid nagu teksti pikkus, lõpmatus regulaaravaldistes ja peaaegu vastavate tekstide genereerimise võimalus.

Lahendus põhineb deterministlike lõplike automaatide lühimate teekondade arvutamisel, et saavutada automaadi servade paaride täielik katmine.

Töö tulemused näitavad, et loodud lahendus suudab genereerida kõik vajalikud tekstid vastavalt regulaaravaldistele ning vastab püstitatud nõuetele. Samuti kirjeldatakse tulemuste valideerimise protsessi, mis hõlmab nii manuaalseid kui ka ühikteste, et kontrollida väljundi täpsust ja süsteemi toimivust erinevates olukordades.

Töös tuuakse ka esile olulisi jõudlusprobleeme, mis võivad tekkida NFA teisendamisel DFA-ks suuremate automaatide korral. Lisaks käsitletakse servade paaride genereerimisel ilmnevaid erandjuhtumeid, mis võivad mõjutada genereeritud tekstide katvust.

Lõputöö pakub põhjalikku ülevaadet tekstide genereerimise meetoditest regulaaravaldiste alusel ning esitab praktilise lahenduse selle protsessi automatiseerimiseks. Töö toob esile ka olulised optimeerimise ja jõudluse parandamise aspektid, mis võivad olla kasulikud edasiste uurimuste ja rakenduste jaoks selles valdkonnas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 5 peatükki, 21 joonist, 5 tabelit.

Abstract

Generating Texts Based on Regular Expressions

This thesis explores methods for generating texts from regular expressions needed for regular expression engines benchmaking, focusing on enhancing the Microsoft Automata library and its Rex subproject. The aim of the thesis is to create a solution for generating texts while considering various constraints such as text length, infinity in regular expressions, and the possibility of generating almost matching texts.

The solution is based on the calculation of the shortest paths and deterministic finite automata in order to achieve complete edge pairs coverage.

The results of the study show that the developed solution can generate all necessary texts according to the regular expressions and meets the set requirements. The thesis also describes the process of validating the results, which includes both manual and unit tests to verify the accuracy of the output and the system's performance in different scenarios.

The thesis highlights significant performance issues that may arise during the conversion of NFA to DFA for larger automata. Additionally, it addresses exceptional cases that may occur during the generation of edge pairs, which can affect the coverage of the generated texts.

The thesis provides a comprehensive overview of methods for generating texts from regular expressions and presents a practical solution for automating this process. It also highlights important aspects of optimization and performance improvement, which can be beneficial for future research and applications in this field.

The thesis is in estonian and contains 33 pages of text, 5 chapters, 21 figures, 5 tables.

Lühendite ja mõistete sõnastik

DFA	*Deterministic finite automata,* deterministlik lõplik automaat
NFA	*Nondeterministic finite automata,* mittedeterministlik lõplik automaat
SFA	*Symbolic finite automata,* sümbolautomaat
URL	*Uniform Resource Locator*, internetiaadress
CPU	*Central Processing Unit,* protsessor
UTF	*Unicode transformation format*
DSFA	*Deterministic symbolic finite automata*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Regulaaravaldised on võimas, paindlik ja efektiivne tekstitöötluse vahend. Regulaaravaldised ise, üldise mustrina on peaaegu nagu väike programmeerimiskeel, võimaldades töödelda teksti. [1]

Regulaaravaldisi kasutatakse erinevate ülesannete lahendamiseks: tekstist andmete ekstraheerimiseks, kasutaja sisestatud andmete valideerimiseks, logide analüüsimiseks, andmeanalüüsi teostamiseks, võrgupakettide sisu kontrolllimiseks jne.

Regulaaravaldiste mootorid on olulised tarkvarakomponendid, mida saab kasutada näiteks Java programmeerimiskeeles, .NET raamistikul ja Perl programmeerimiskeeles. Mõned näited regulaaravaldiste mootoritest on Re2 ja Hyperscan. Regulaaravaldised põhinevad matemaatilistel mudelitel, tavapäraselt automaatide teoorial nagu deterministlikud lõplikud automaadid (DFA) ja mittedeterministlikud lõplikud automaadid (NFA). Nende keerukus võib olla äärmiselt kriitiline, kuna halvasti optimeeritud regulaaravaldiste abil on võimalik teostada ründeid, näiteks *Regular expression Denial of Service* (ReDoS) rünnakuid. Ründaja võib olla võimeline keerulise regulaaravaldisega käivitama protsessi, millega kurnab ohvri CPU ressursse ja põhjustab oma tegevusega teenuse keelamise. [2] Seega regulaaravaldiste mootorite jõudlus on oluline.

Tekib küsimus, kuidas hinnata regulaaravaldiste mootorite jõudlust ja võrrelda neid teiste mootoritega. Jõudluse hindamiseks kasutatakse tavaliselt *benchmarke*, ehk võrdlusanalüüsi. Kuigi *benchmark*'id peaksid koosnema nii regulaaravaldistest kui ka tekstidest, tihtipeale koosnevad nad ainult regulaaravaldistest. Isegi kui nad sisaldavad ka tekste, need tekstid on väga spetsiifilised. Näiteks, Mark Twaini tekstide kogust [3], mida tihti kasutatakse regulaaravaldiste *benchmark*'ina, ei ole kasu veebirakenduste arendajatele, kes valideerivad regulaaravaldiste abil kasutaja sisendit veebilehel. Seda enam tihti pole võimalik pärisandmeid avalikustada (nt. serveri logide või kasutaja

isikuandmete puhul). Seega *benchmark*'ide kvaliteeti tõstmiseks on vaja tekstide generaatorit.

Regulaaravaldiste mootorite jõudluse võrdlemiseks ei ole alati vaja genereerida kõiki võimalikke tekste regulaaravaldise põhjal. Jõudluse hindamiseks on tavaliselt vaja testida mootorit erinevates olukordades. Seega on oluline luua valik tekste, mis katavad erinevaid mustreid ja keerukusastmeid, kuid mitte tingimata kõiki võimalikke variante. Selline lähenemine on tõhusam ja ajasäästlikum, kuna see võimaldab keskenduda olulistele testjuhtudele, mis toovad esile mootorite erinevused jõudluses. On lahendusi, mis genereerivad üks või mitu teksti vastavalt regulaaravaldisele (näiteks Browserling *Text from regex* [4]). Samuti on lahendusi, mis genereerivad kõikvõimalikud vasted regulaaravaldisele nagu Wimpy Programmeri *regex-to-strings* [5].

Olemasolevaid lahendusi on erinevaid, aga see lõputöö keskendub Microsoft Automata teegile. See on Microsoft Research poolt arendatud projekt. Automata on .NET-i teek, mis pakub algoritme regulaaravaldiste, automaatide ja muundurite koostamiseks ja analüüsimiseks. Kuigi antud teegiga on regulaaravaldiste põhjal võimalik genereerida mitu teksti, vajame siiski rohkem kontrolli genereerimisprotsessi üle. Seega on vaja luua parametriseeritud genereerimislahenduse, mis võimaldaks genereerida mitmeid tekste, arvestades vajalikku pikkust, sümboli korduste arvu ning kas genereeritav tekst peaks vastama etteantud regulaaravaldisele.

Selle bakalaureusetöö raames on eesmärk täiustada Microsoft Automata teeki nii, et saaks tekstide genereerimisprotsessi parametriseerida.

Oodatud tulemusena peaks tekstide genereerimisalgoritm toetama järgmiseid funktsionaalsusi:

- Erinevate tekstide genereerimine regulaaravaldiste põhjal
- Tekstide genereerimist lisanduvate parameetrite näol - pikkuste vahemik, sümboli korduste arv
- Peaaegu vastavate tekstide genereerimist regulaaravaldiste põhjal

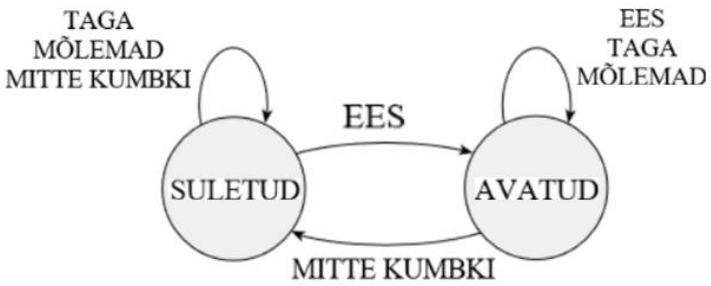
2 Taust ja metoodika

See peatükk kirjeldab automaatide teooriat, töö protsessi ning kasutatud tööriistasid.

2.1 Automaatide teooria

Automaatide teooria käsitleb arvutuslike matemaatiliste mudelite definitsioone ja omadusi. Need mudelid mängivad rolli mitmes arvutiteaduse rakendusvaldkonnas. Ühte mudelit, mida nimetatakse lõplikuks automaadiks, kasutatakse tekstitöötluses, kompilaatorites ja riistvara kujundamisel. [6]

Automaatne ukse kontroller on hea näide lõplikust automaadist. Automaatsed uksed avanevad sageli kaubanduskeskuste sisse- ja väljapääsude juures, kui kontroller tajub, et inimene läheneb. Automaatuksel on ees ala, mis tuvastab ukseavast läbi astuva inimese kohalolu. Teine ala asub ukseava taga, et kontroller saaks ust lahti hoida piisavalt kaua, et inimene saaks sellest lõpuni läbi minna, ning et uks ei tabaks avanedes kedagi, kes selle taga seisab. Kontroller saab olla kahes olekus, kas "AVATUD" või "SULETUD", mis tähistab ukse vastavat seisundit. Nagu on näidatud järgmistel joonistel, on neli võimalikku sisendit: "EES" (see tähendab, et isik seisab ukseava ees oleval alal), "TAGA" (see tähendab, et isik seisab ukseava taga asuval alal), "MÕLEMAD" (see tähendab, et inimesed seisavad mõlemal pool ust) ja „MITTE KUMBKI” (see tähendab, et keegi ei seisa kummalgi pool ust) [7].



Joonis 1: Automaatse ukse kontrolleri olekudiagramm. [7]

Olek\sisend	MITTE KUMBKI	EES	TAGA	MÕLEMAD
SULETUD	SULETUD	AVATUD	SULETUD	SULETUD
AVATUD	SULETUD	AVATUD	AVATUD	AVATUD

Tabel 1: Automaatse ukse kontrolleri oleku üleminekutabel. [7]

Kontroller liigub olekust olekusse, sõltuvalt vastuvõetavast sisendist. Kui see on SULETUD olekus ja võtab vastu sisendit EI KUMBKI või TAGA, jääb see SULETUD olekusse. Lisaks, kui sisend MÕLEMAD on vastu võetud, jääb see SULETUD olekusse, kuna ukse avamisel on oht. Kui aga saabub sisend EES, liigub see AVATUD olekusse. AVATUD olekus, kui võetakse vastu sisend EES, TAGA, või MÕLEMAD, jääb see AVATUD olekusse. Kui sisendiks on EI KUMBKI, naaseb SULETUD olekusse.

Lõplikud automaadid liiguvad olekust olekusse, võttes vastu sisendit ja järgides reegleid vastavalt sellele, millises olekus nad asuvad ja millist sisendit nad saavad. Matemaatiliselt määratletakse lõplikku automaati 5-korteežina, mis koosneb olekutest, sisendi tähestikust, üleminekufunktsioonist, algolekust ja aktsepteeritud olekute hulgast. Üleminekufunktsioon kirjeldab, kuidas automaat liigub ühest olekust teise, võttes arvesse antud sisendit. Lõplikud automaadid võivad olla kas deterministlikud (DFA) või mitte-deterministlikud (NFA). DFA puhul on igale sisendile määratud täpselt üks järgmine olek, samas kui NFA-s võib ühe sisendi kohta olla mitu võimalikku järgmist olekut. Automaat võib sisaldada tühja väärtust tähistavat ε käike. ε ei ole automaadi tähestiku täht, vaid tähistab tühja väärtust, stringi pikkusega 0.

Deterministliku lõpliku automaadi formaalne definitsioon [7]:

Deterministlik lõplik automaat on viiene korteež $(Q, \Sigma, \delta, q0, F)$, kus

1. Q on lõplik hulk nimega olekud
2. Σ on lõplik hulk nimega tähestik
3. $\delta : Q \times \Sigma \longrightarrow Q$ on üleminekufunktsioon
4. $q0 \in Q$ on algolek
5. $F \subseteq Q$ on hulk aktsepteerituid olekuid

Vaatame näidet lõplikust automaadist nimega M1, millel on kaks olekut. Kujutame seda ette kui süsteemi, mis võib olla ühes kahest olekust ja mis muudab oma olekut vastavalt sisenditele.

Formaalne kirjeldus automaadist M1 on järgmine: ({q1, q2}, {0, 1}, δ, q1, {q2}). Seda kirjeldust saab lahti seletada järgmiselt:

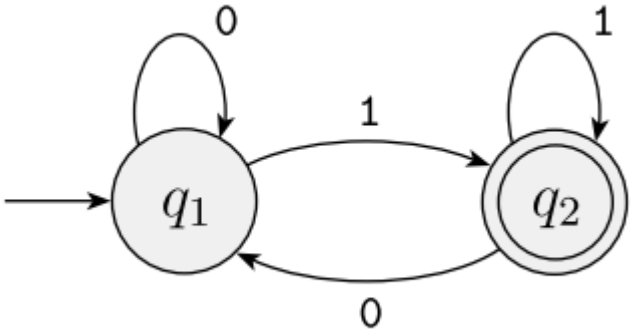
- {q1, q2} - hulk kõikidest võimalikest olekutest, milles automaat võib olla. Selles näites on kaks olekut: q1 ja q2.
- {0, 1} - sisendite hulk, mida automaat võib vastu võtta. Selles näites on kaks võimalikku sisendit: 0 ja 1.
- δ - see on üleminekufunktsioon, mis määrab ära, kuidas automaat ühest olekust teise liigub, sõltuvalt sisendist.
- q1 - algolek, milles automaat alustab oma tööd.
- {q2} - hulk lõppolekutest. Automaat loetakse vastuvõtlikuks, kui ta lõpetab oma töö ühes nendest olekutest.

Üleminekufunktsioon δ määrab, kuidas automaat liigub ühest olekust teise, sõltuvalt saadud sisendist. Üleminekufunktsioon esitatud tabelis 2.

	0	1
q1	q1	q2
q2	q1	q2

Tabel 2: Näide kahe olekuga DFA M1 üleminekufunktsioonist tabeli kujul. [7]

Automaat M1 olekudiagrammina on joonisel 2.



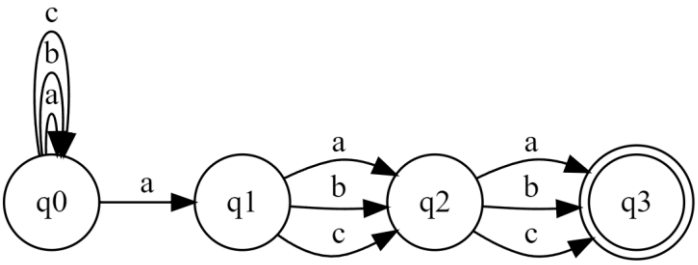
Joonis 2: Näide kahe olekuga DFA-st M1. [7]

Vaatamata lõplike automaatide laiaulatuslikule rakendusele on neil mudelitel suur puudus: kõige tavalisemates vormides saavad nad hakkama ainult lõplike ja väikese tähestikuga. Selle piirangu ületamiseks võimaldavad lõplikud sümbol automaadid üleminekutel kanda predikaate ja funktsioone, ning seetõttu laiendada lõplikke automaate, et töötada üle lõpmatu tähestiku. [8]

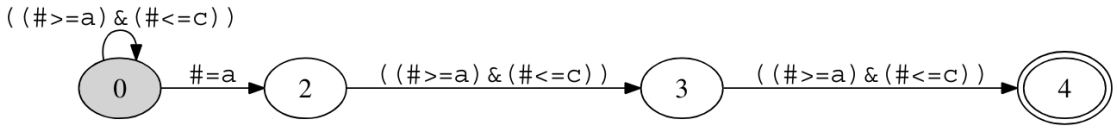
Lõplikuid sümbolautomaate nimetatakse SFA-deks (*Symbolic Finite Automata*). SFA-s on üleminekud olekute vahel märgistatud valemitega või predikaatidega, mitte üksikute sümbolitega. [9] SFA-d on tavaliselt väiksemad ja efektiivsemad, kuna need võimaldavad üleminekuid mitme tähe kombinatsioonidel, kasutades predikaate. See tähendab, et SFA võib sageli esitada keerukamaid automaate kompaktsemalt kui DFA. SFA võib olla deterministlik või mittedeterministlik. [10]

Lõplike automaatide olekute arv on oluline mõõdik automaadi keerukuse ja selle poolt lahendatava probleemi keerukuse hindamiseks. Rohkem olekuid tähendab keerukamat automaati, sest sellele läheb rohkem mälu ja töötlemise ressurssi.

Toon siinkohal näite deterministlikust SFA-st (joonisel 4) võrreldes DFA-ga (joonisel 3). Näiteks regulaaravaldise „[a-c]*a[a-c]{2}“ SFA ja DFA esitus on erinev. SFA on sageli kompaktsem, kuna see suudab käsitleda üleminekuid nii 'a', 'b' kui 'c' jaoks ühe üleminekuga, samas kui DFA peab iga tähte eraldiseisvalt võtma, mille tulemuseks on rohkem olekuid ja üleminekuid. DFA-l on rohkem olekuid ja keerukamad üleminekud, samas kui SFA suudab lihtsamalt ja efektiivsemalt hallata sama keelt.



Joonis 3: Näide DFA kohta.



Joonis 4: Näide DSFA kohta. [9]

2.2 Objekti kirjeldus

Bakalaureusetöös tegeletakse Microsoft Automata teegiga ja selle alaprojektiga Rex. Microsoft Automata on teek, mis pakub algoritme regulaaravaldiste, automaatide ja muundurite koostamiseks ja analüüsimiseks [14]. Rex on selle alamprojekt, mis võimaldab regulaaravaldiste põhjal tekste genereerida [17]. Töö käigus täiendati Microsoft Automata teeki nii, et see suudaks genereerida rohkem tekste ning võimaldaks tekstide genereerimisprotsessi parametriseerida.

Loodud lahendus on arendatud .NET raamistikul ja C# programmeerimiskeeles.

2.3 Töö protsess ja tööriistad

2.3.1 Töö protsess

Eesmärgi saavutamiseks jälgitakse tarkvaraarendusmetoodikat ja jagatakse protsess kolmeks etapiks:

1. Probleemi ja olemasolevate lahenduste analüüs

2. Algoritmi arendus ja testimine tulemuste valideerimiseks

3. Tulemuste analüüs

2.3.2 Tööriistad

Arendus on tehtud .NET raamistikuga, sest MS Automata on kirjutatud .NET raamistikuga. Arenduse ehitame selle teegi peale, kasutades olemasolevat funktsionaalsust.

Tööprotsessi ja selguse huvides tuli vajadus MS Automata genereeritava automaadi graafilise esituse järele. Teegis on võimalus viia automaat .dot faili formaati. DOT on graafi kirjeldav keel. Kasutades kaasjuhendaja arendatud automaadi visualiseerimise tööriista [11], mis tarbib sisendina .dot/graphviz formaati, sain hõlpsasti ja kiirelt ülevaate automaadist, mille teek genereeris. Seda saab kasutada kui DFA servade arv ei ole liiga suur. Samuti kasutasin automaatide visualiseerimiseks ka ühte teist dot formaati toetavat automaadi visualiseerijat [12], et kuvada automaate valgel taustal.

3 Töö tulemused

Kolmandas peatükis keskendutakse juba olemasolevate lahenduste tutvustamisele, valitud teegi (mida hakatakse täiendama) võimalustele ja piirangutele, nõuete seadmisele, lahenduse kirjeldusele, implementeerimisele, optimeerimisele ning valideerimisele.

3.1 Olemasolevad lahendused

Allikatest, nagu Maxim Kulkini Hypothesis-regex GitHubi projekt [13], Browserlingi *Text from regex* tööriist [4], Wimpy Programmeri *regex-to-strings* GitHubi projekt [5], AutomataDotNeti GitHubi projekt [14] ning Arext projekt [15] on välja töötatud mitmeid lahendusi, mis võimaldavad genereerida tekste etteantud regulaaravaldise põhjal.

Hypothesis [16] on teek ühiktestide ja testandmete genereerimiseks, seal hulgas ka teksti genereerimiseks regulaaravaldise põhjal. Selle jaoks on eraldi alamprojekt [13]. Projekt on piiratud funktsionaalsusega, võimaldades genereerida vaid ootamatuid tihtipeale loetamatuid tekste. See ei taotle luua kena väljanägemisega stringe, vaid selle asemel mistahes hullumeelset ootamatut kombinatsiooni, mis sobib antud regulaaravaldisega. Kasutajal puudub kontroll genereeritava väljundi osas. Lisaks on seda viimane kord uuendatud 6 aastat tagasi.

Browserlingi Text from regex [4] on sarnane veebipõhine tekstide generaator regulaaravaldise põhjal. See võib genereerida mitu vastavat suvalist teksti regulaaravaldise põhjal, aga mitte kõiki variante. See on algeline ja ei rahulda nõudeid, mis selle töö raames on püstitatud nagu näiteks teksti pikkus ja peaaegu vastavus.

Wimpy Programmeri *regex-to-strings* [5] on põhjalikum ja omab rohkem funktsionaalsust. See on veebipõhine rakendus, mis loob kõikvõimalikud vasted, saab ka määrata kui palju tekste genereerida. Regulaaravaldises piiramatu arvu sümbolite sobitamises seatakse ülemiseks piiriks 100, et ei tekiks lõpmatut tsüklit.

Üks olemasolevatest tööriistadest on Arext, mis on automaatne regulaaravaldiste testimise tööriist, mis põhineb regulaaravaldise täielikul katmisel tekstide genereerimisel. [15] See põhineb Google poolt arendatud RE2 teegil, mis kasutab taustal peamiselt deterministlike lõplike automaate. Lisaks ei toeta see UTF-16 kodeeringut, toetades ainult UTF-8 kodeeringut. RE2 töötab baitide võrdlemise tulemusena. Kasutades Graphviz saab

Arext visualiseerida regulaaravaldisi DFA graafina aga kuna see on baitide põhine, siis graafikult ei ole sisendid ehk käigud loetavad. Arext ei genereeri kõiki tekste. Selle asemel genereerib tekste eemärgiga saavutada täielik servapaaride katvus.

Veel üks lahendustest on Microsoft Researchi projekt Rex [17]. Automata on .NET-i teek, mis pakub algoritme regulaaravaldiste, automaatide ja muundurite koostamiseks ja analüüsimiseks. Lisaks klassikalistele sõnapõhistele automaatidele sisaldab see ka puustruktuuriga automaate ja puustruktuuriga muundurite analüüsi algoritme. Teek hõlmab nii piiratud tähestike algoritme kui ka nende sümboolseid vasteid. Sümbolautomaatides on konkreetsed märgid asendatud märgipredikaatidega. Sellised predikaadid võivad ulatuda väga suurte või isegi lõpmatute tähestike, näiteks täisarvude vahel. [18] Microsoft Automata teegis tõlgitakse taustal regulaaravaldisi lõplikeks automaatideks sümbolautomaatide esituses ehk see põhineb lõplikel sümbolautomaatidel. Rex on osa Microsoft Automata teegist. See on tööriist, mis võimaldab tõhusalt genereerida vastavaid tekste regulaaravaldiste põhjal. Teeki on lihtne kasutada, kuid see ei genereeri kõiki teksti variante. Saab määrata mitu teksti see tagastab.

	Avatud lähtekoodiga	Kõikvõimalike tekstide genereerimine	Peaaegu vastavate tekstide genereerimine	Automaadi visualiseerimine (.dot faili loomine)
Hypothesis	Jah	Ei	Ei	Ei
Browserling	Ei	Ei	Ei	Ei
Wimpy Programmer	Jah	Jah	Ei	Ei
MS Automata	Jah	Ei	Ei	Jah
Arext	Jah	Ei	Ei	Jah

Tabel 3: Olemasolevate lahenduste võrdlus.

Olemasolevaid lahendusi on mitmeid, kuid paraku reaalsuses puudub hea lahendus stringide genereerimiseks regulaaravaldiste põhjal. Olemasolevate lahenduste puudus on ebapiisav kontroll genereeritavate tekstide üle, st algoritmid ei ole parametriseeritud. Olemasolevate lahenduste võrdlust saab näha tabelist 3.

Olemasolevate lahenduste peamiseks kitsaskohaks on nende funktsionaalsuse piiratus, kuna need suudavad genereerida vaid täpseid vasteid ning ei võimalda luua tekste, mis vastaksid antud omadustele. Näiteks, *benchmarkide* koostamisel võib olla vajadus nii täpsete vastete kui ka ligikaudu vastavate tekstide järele vastavalt regulaaravaldisele. Samuti võib teksti pikkus mõjutada süsteemi jõudlust. Lisaks sellele on mõned neist teekidest viimati uuendatud mitu aastat tagasi, mis võib viidata nende vananenud olekule.
Tekstide genereerimise kontekstis *benchmark*'ide jaoks ei ole otstarbekas genereerida kõikvõimalikke tekste, mida konkreetse regulaaravaldise põhjal saab luua. Pole vajadust testida kõiki võimalikke teksti variante, sest vasteid võib olla liiga palju ja nende testide jooksutamine võtaks üleliia aega. Tähtsam on katta ära automaadis võimalikud olekud ja nende üleminekud. Samuti ei ole seal võimalust genereerida peaaegu vastatavaid tekste.

3.2 Microsoft Automata Rex

Valisime oma probleemi lahendamiseks MS Automata teegi edasi arendamise, sest see põhineb sümbolautomaatidel, mis teeb selle kiireks. Veel on see hea teek, kus on palju automaatidel põhinevaid algoritme. Lisaks on Automata tuletistel põhinev ja tuletistega loodud automaat on reeglina paremini minimeeritud kui ilma. [19] Kui on ette antud mitu regulaaravaldist, saab Rex-i kasutada liikmete genereerimiseks, mis vastavad kõigile neile. Vaikimisi eeldab Rex UTF-16 kodeeringut, kuid seda saab piirata ka ASCII kodeeringuga. [17] Peamine põhjus, miks sai selle teegi täiendamise kasuks otsustatud oli see, et MS Automata on põhineb sümbolautomaatidel, mis on efektiivne võrreldes DFA-ga. Rex on võimeline käsitlema suuri regulaaravaldisi, mille automaadil on sadu või isegi tuhandeid olekuid sageli alla sekundi. See näitab kui tõhus on sümbolautomaatidel põhinev MS Automata.

Antud raamistik pakub laiaulatuslikku tööriistakomplekti regulaaravaldiste SFA-deks teisendamiseks, nende manipuleerimiseks ja saadud automaatide visualiseerimiseks.

Kuigi see hõlmab laia valikut regulaaravaldiste konstruktsioone, ei ole mõned funktsioonid veel toetatud.

Automata projekti alamprojekt on Rex, mis lisab regulaaravaldise liikmete genereerimise funktsioonid. Rex kasutab Automata teegi funktsioone nagu sümboolsete lõplike automaatide (SFA) loomine, SFA-de minimeerimine, NFA determineerimine, SFA-de kui suunatud graafide visualiseerimine.

3.2.1 Microsoft Automata Rex võimalused

Automata Rex kasutamise võimalused:

1. Rex aktsepteerib regulaaravaldisi sisendina. See teisendab need regulaaravaldised vastavateks SFA-deks, kasutades teisendusalgoritme.

2. Rex võimaldab genereerida juhuslikke stringe, mis vastavad antud regulaaravaldisele. See kasutab genereerimisprotsessi juhtimiseks SFA-sid. Protsess hõlmab automaadi läbimist algolekust üleminekute alusel kuni lõppseisundini. Igal etapil mõjutab üleminekuvalikuid juhuslikkus, tagades, et genereeritud stringid on juhuslikud, kuid kehtivad.

    - Rex suudab genereerida kasutaja etteantud arvu unikaalseid tekste, mis vastavad kõikidele sisendina antud regulaaravaldistele.

    - Rex genereerib regulaaravaldise põhjal teksti, mis sisaldab seda mustrit. Kui tahta teksti, mis vastaks tervenisti antud mustrile, tuleb kasutada regulaaravaldise alguses sümbolit ‘^’ ja lõpus sümbolit ‘$’, mis ütlevad ära, kuidas vaste peab algama ja lõppema.

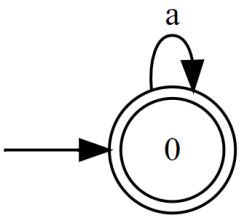
    - Näiteks 3 unikaalse teksti genereerimiseks "^Huck[a-zA-Z]+|Saw[a-zA-Z]+$" regulaaravaldise põhjal, võib tagastada see näiteks sellise tulemuse: “瘋準SawezvyJz”, “SawZu”, “Huckg”.

    - Kui sisendina anda mitu regulaaravaldist, siis Rex leiab nende ühendi. Regulaaravaldise "^[1-9].{1}$" ja "^.{1}[abc]$" korral genereeritakse näiteks sellised tekstid: “5c”, “2a”, “9a”.

3. Lisaks juhuslikule genereerimisele toetab Rex *stringide* ühtlast genereerimist. See tähendab, et igal automaadi poolt aktsepteeritud *stringil* on võrdne tõenäosus genereerida. See tagab genereeritud stringide jaotamise õigluse. Automata Rex toetab regulaaravaldise omadusi nagu: '.', '^', '$', '[…]', '[^…]', '*', '+', '?', vahemik {min, max}, grupeerimine '(…)', '|', '\w', '\W'.

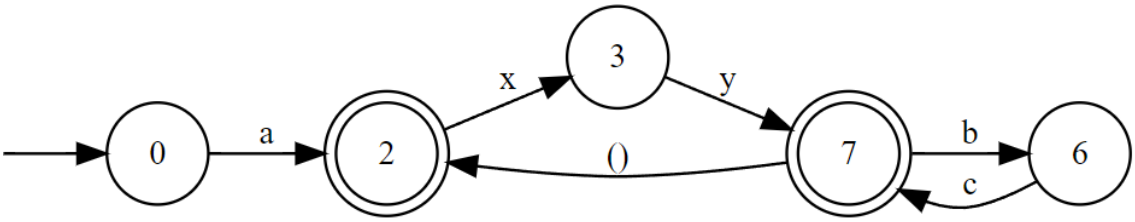
Testimiseks mõeldud tekstide genereerimisel meid huvitab, kuidas töötab metamärk '*', mis leiab vasteks null või rohkem sümbolit ning '+', mis leiab vasteks üks või rohkem sümbolit. MS Automata Rex genereerib sellisel juhul automaadi, kus esineb tsükleid. Paremaks mõistmiseks vaatame paari näidet.

Regulaaravaldise üksiku sümboli piiramatu arvu märkide sobitamiseks näiteks "a*" korral luuakse selline automaat, mis sisaldab tsüklit iseendasse:



Joonis 5: Automaat sisaldab tsüklit iseendasse.

Regulaaravaldise grupeeritud sümbolite piiramatu arvu korduste leidmiseks näiteks "a(xy(bc)*)*" korral luuakse selline automaat, mis sisaldab tsükleid:



Joonis 6: Automaat sisaldab tsüklit mitme oleku vahel.

3.2.2 Microsoft Automata teegi piirangud

Automata toetab paljusid klassikalisi regulaaravaldiste mootorite omadusi, kuid on ka funktsionaalsust, mida see ei toeta ehk ei saa etteantud regulaaravaldise süntaksist aru. MS Automata ei toeta: tagasiviiteid (*references*) '\1' (vastavad tekstile, mis on varasemalt vaste leidnud); '\b' (võimaldab otsida ühte tervet sõna), '\B' (vastab tühjale

stringile, mis ei ole alguses ega lõpus); ahned konstruktsioonid nagu (?>) ja (?<); ennetavad konstruktsioonid (?!) ja (?<!); nõutavad konstruktsioonid (?=) ja (?<=); testimisgrupid (*test construct*) nagu (?(...) |); testimisviited (*test construct*) nagu (?(n) |); '\G' on globaalse otsimise jaoks. Mittetoetatud konstruktsioonide kasutamisel annab teek vastava veateate. MS Automata ei toeta ka *lookaround*'e. Regulaaravaldise *lookaround* võimaldab sobitada mustrit ainult siis, kui sellele järgneb või eelneb mõni muu muster.

3.3 Nõuded

Regulaaravaldiste mootorite jõudluse võrdlemiseks vajaminev testandmete kogu ei tohiks olla üleliia suur, sest see võtaks ajaliselt liiga kaua aega. Tegelikkuses ei ole vajadust testida kõiki tekstide variante. Regulaaravaldise „[1-3][a-c]" korral pole oluline testida iga üksikut sümbolit 1, 2, 3 ja a, b, c eraldi, sest meid huvitab pigem nende vahemik. Seega, selle asemel, et luua tekste, mis sisaldavad kõiki võimalikke kombinatsioone nagu "1a", "1b", "1c", "2a", "2b", "2c", "3a", "3b", "3c" piisab eesmärgi saavutamiseks tekstidest, mis katab kõik need tähtede vahemikud ja nende kombinatsioonid. See lähenemine võimaldab meil vähendada testandmete hulka suurelt, kuid samas säilitada katvust ja tähenduslikkust. Jõudlust mõjutab enamasti kõige rohkem automaadi olekute arv ja konfiguratsioon. On vajadus valida vahemikke ehk automaadi servasid mitte konkreetseid sümboleid. Mõistlik on olekute ja servade katmine testandmete loomise kontekstis. See vähendab genereeritavate tekstide arvu oluliselt.

Arext regulaaravaldiste testimise projektist inspireeritult olen otsustanud tekstide genereerimise probleemi lahendamiseks võtta automaadi servade paaride 100% katmise. Arext projektis implementeeritakse samasugust ideed. Seal kasutatakse Google väljatöötatud regulaaravaldiste mootorit Re2 selleks, et luua vastav DFA antud regulaaravaldisele. Arext projektis lahendatakse servade paaride algoritm baitidega.

Selle lahenduse põhiline idee seisneb kõikvõimalike teksti konfiguratsioonide kombinatsioonide genereerimises. See seisneb DFA graafi kõikide servade paaride leidmisel ja lühimate teede arvutamisel.

Olemasolevate lahenduste analüüsi, Microsoft Automata Rex teegi võimaluste ja piirangute põhjal ning arvestades vajalikku tekstide katmist, töötati välja järgmised nõuded. Algoritm peab toetama:

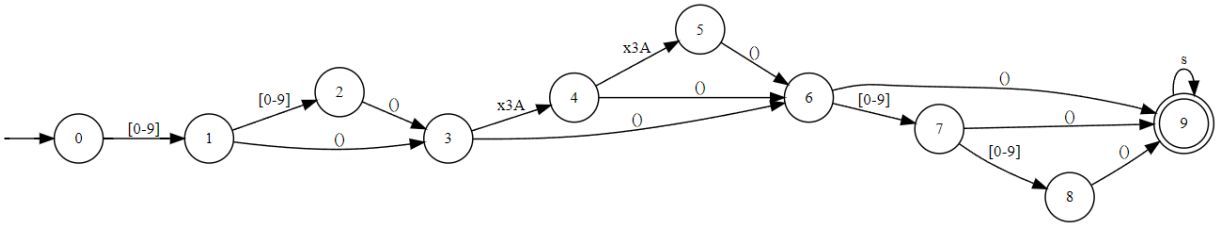
- Erinevate vajalike tekstide genereerimist, mis kataksid ära automaadi kõik olekud ja servade paarid
- Genereeritava teksti minimaalse ja maksimaalse pikkuse määramist
- Peaaegu vastavate tekstide genereerimist
- Regulaaravaldises sümbolite korduste arvu määramist

3.4 Algoritmi kirjeldus

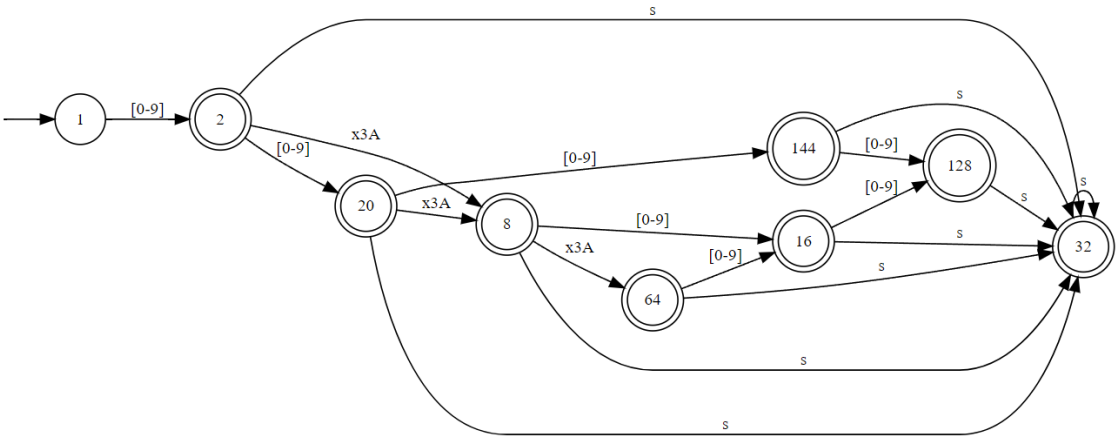
Alguses realiseerisin algoritmi Automata genereeritud SFA põhjal, mis oli sisuliselt NFA, see tähendab, et see automaat sisaldas tühjasid ehk epsilon käike. Algoritmi põhiidee – katta ära kõik servade paarid automaadis, töötab ainult DFA puhul, sest NFA puhul võib ühest automaadi olekust väljuda mitu serva. See tähendab, et algoritm ei pruugi leida kõiki teekondi automaadis, sest see põhineb lühimate teede kasutamisel. Epsiloniga transaktsioonid tekitavad nö otseteid, mis on lühemad. Seega võib tekkid olukord, kus algoritm ei leia kõige pikemat teksti, mis vastab antud regulaaravaldisele. Kõikide teekondade leidmine NFA põhjal on keeruline.

Toon näiteks sellise regulaaravaldise: "([0-9]{1,2}[:]{0,2}[0-9]{0,2}[\\s]{0,})"

Automata genereerib sellele vastava SFA, mis on sisuliselt NFA. Algoritm genereerib 13 unikaalset teekonda. Automata teegis saab teisendada mittedeterminstliku lõpliku automaadi deterministlikuks. DFA põhjal genereerib algoritm lausa 34 unikaalset teekonda.



Joonis 7: Regulaaravaldis NFA-na.



Joonis 8: Regulaaravaldis DFA-na.

Üheks nõudeks on sümboli korduste arvu määramine. Selleks, et otsustada kui palju kordusi metamärkkide '*' ja '+' korral võtta, on kõige lihtsam asendada regulaaravaldises need sümbolid vahemikuga "{}". Paraku muutub selle võrra ka DFA keerukamaks. Kasutaja mugavuse huvides, saab ta kasutada neid sümboleid ja määrata nende korduse arvu, kuid seda saab ka sisendina antavas regulaaravaldises vahemiku näol edastada.

Sageli kasutatakse testandmete kvaliteedi mõõtmiseks katvuse kriteeriume. Koodi katvuse kriteeriumid ei ole sobivad regulaaravaldiste jaoks. Wang ja Stolee pakkusid välja katvuse DFA-esitusel põhinevate regulaaravaldiste mõõdikud, sealhulgas kolm katvuse taset: *Node Coverage* (NC), *Edge Coverage* (EC) ja *Edge-Pair Coverage* (EPC). [15] Algoritmi eesmärk on saavutada 100% sõlmede, servade ja servapaaride katvus.

Algoritmi põhiline idee seisneb kõikide võimalike tekstikonfiguratsioonide genereerimises, mis vastavad etteantud piirangutele ja sobivad antud regulaaravaldiste vasteteks. Selleks kasutatakse DFA-d, mis on loodud regulaaravaldistest, ja algoritmi, mis arvutab lühimad teekonnad DFA olekute vahel, et tagada täielik katvus.

Algoritmi implementeerimine hõlmab kolme põhietappi:

1) Automaadi loomine ja determineerimine

DFA loomine regulaaravaldistest: Regulaaravaldised muudetakse DFA-ks, võimaldades leida olekute vajalikke teekondi. MS Automata RexEngine loob sisendina antud regulaaravaldiste põhjal SFA, mis sisaldab tühjasid ehk epsilon servi, millega ei saa leida

kõiki teekondi. See on põhjus, miks ma oma lahenduse realiseerimisel teisendan SFA DFA-ks. Metamärkide nagu '*' ja '+' haldamiseks asendan need regulaaravaldises vahemikuga, et kasutaja saaks määrata korduste arvu. Vaikimisi võetakse vahemikud vastavalt '{0,1}' ja '{1}'.

2) Vajalike teksti konfiguratsioonide genereerimine

Lühimate Teekondade Arvutamine: DFA olekute vahel arvutatakse lühimad teekonnad, mis võimaldavad genereerida katvate tekstide komplekti.

a. Lühimate teekondade arvutamine algolekust kõikidesse teistesse olekutesse

b. Lühimate teekondade leidmine kõikidest olekutest kõikidesse lõppolekutesse kasutades laiuti otsingu algoritmi

c. DFA kõikide olekute paaride genereerimine

d. Kõikidele olekute paaridele moodustatakse automaadis teekonnad kasutades lühimaid teekondi algolekust kuni lõppolekuni

3) Tekstide genereerimine

Põhinedes arvutatud teekondadele, leitakse iga olekute paarile ehk servale sobivate sümbolite vahemikud, millest valitakse suvaliselt üks sümbol, võttes arvesse ka etteantud piiranguid. Genereeritava teksti loetavuse tõstmiseks eelistab algoritm tähti ja numbreid. Võimalusel eelistatakse sümboleid *unicode*'ga 32-127.

3.5 Algoritmi realiseerimine

Lahenduse realiseerimisel lõin mitmeid meetodeid ja funktsionaalsusi [20].

Meetodi *GenerateTestStrings* eesmärk on luua testimiseks vajalik stringide kogum, mis vastab antud regulaaravaldistele ning samal ajal arvestab erinevaid piiranguid, näiteks teksti pikkus ja võimalikud kordused regulaaravaldistes. Parameetrite hulka kuuluvad regulaaravaldised, peaaegu vastavate tekstide loomise võimalus ning tärni (*) märgi korduste minimaalne ja maksimaalne arv (+ sümboliga toimib samamoodi). Meetod töötab järgmiselt: alguses luuakse uus RexEngine eksemplar, mis võimaldab luua regulaaravaldistest SFA. Seejärel kohandatakse antud regulaaravaldised vastavalt tärni

korduste piirangutele ning loodud SFA muudetakse deterministlikuks ehk DFA-ks, sest algoritmi idee põhineb just deterministlikul lõplikul automaadil. Edasi arvutatakse DFA olekute vahel lühimad teekonnad, võttes vajadusel arvesse peaaegu vastavate stringide loomist. Võib juhtuda, et DFA olekute vahel on mitu serva, mõlema serva sümbolite vahemike liitmiseks tagastab *ComputeShortestPaths* meetod olekute paarid, et seejärel leida sobivad sümbolid. Lõpuks genereeritakse stringid ja lisatakse need vastavalt määratud piirangutele väljundisse. Soovitud pikkusega tekstid saadakse kõikide tekstide filtreerimise tulemusena.

GenerateTestStrings meetodi üheks parameetriks on boolean väärtusega `almostMatch`, mis määrab selle, et meetod tagastab antud regulaaravaldisele peaaegu vastavad tekstid. Vaikimisi tagastab meetod sobivaid vasteid. Parameetrid `starMinRepeat` ja `starMaxRepeat` määravad ära mitu kordust võtta '*' ja '+' metamärkide puhul. Põhimõtteliselt on tegemist vahemikuga. Vaikimisi võetakse '*' kordust vahemikus {0,1} ja '+' üks kord kui ei ole määratud teisiti. Parameetrina saab ka kaasa anda teksti pikkuse, `minLength` ja `maxLength` piiravad ära kui pikad tekstid tagastatakse. Meetodi GenerateTestStrings signatuuri näeb jooniselt 9.

```
public List<string> GenerateTestStrings(RegexOptions options, bool almostMatch = false, int starMinRepeat = 0, int starMaxRepeat = 1, int minLength = 0, int maxLength = Int32.MaxValue, params string[] regexes)
```

Joonis 9: Teksti genereerimise meetodi signatuur.

Kui on funktsionaalsus, mis salvestab regulaaravaldised DFA andmete põhjal, saab jätkata stringide komplekti loomisega. Stringide genereerimise protsess koosneb kolmest etapist. Lühima teekonna arvutamine iga oleku jaoks, lühimate teekondade arvutamine iga oleku jaoks igasse lõppolekusse ja *EdgePairPool*'i loomine, eesmärgiga tagada automaadi olekute täielik katvus. *EdgePairPool*'i kasutamine muudab DFA teisendamise sobivateks stringideks kiiremaks. Lühimate teekondade arvutamine, arvestades mis tahes olekut, serva või servade paari, teame kohe lühimat DFA teekonda nende katmiseks. Seega saame hõlpsasti genereerida katvate stringide komplekti.

Funktsioon *ComputeShortestPathsFromInitState*() arvutab lühimad teed algolekust kõigisse teistesse olekutesse ja salvestab need sõnastiku tüüpi andmestruktuuri, kus võtmeks on vastav olek, millesse teekond algolekust viib. Selle tulemusena on igal olekul

teave lühimast teeskonnast algolekust iseendani. Lühimate teede arvutamiseks olekutest lõppolekuteni kasutame funktsiooni *ComputeShortestPathsToFinal*(). MS Automata loodud DFA-l võib olla rohkem kui üks lõppolek. Rakendame funktsiooni kõigile neile lõppolekutele. Lühimad teekonnad lõppolekutesse salvestatakse samuti sõnastiku tüüpi andmestruktuuri, kus võti on vastav olek, kust teekonnad algasid lõppolekutesse. Iga oleku kohta on salvestatud *Dictionary* hoidmaks võtmena lõppoleku väärtust ja teekonda sinna.

Meetod *ComputeShortestPathsFromInitState*() toimib järgmiselt: see initsialiseerib järjekorra andmestruktuuri laiuti otsinguga (BFS) läbimiseks, alustades algolekust. Kui järjekord ei ole tühi, eemaldab see oleku ja uurib selle naabreid. Iga naabri jaoks, keda varem pole külastatud, loob see uue lühima tee algolekust selle naabri juurde ja seab naabri edasiseks uurimiseks järjekorda. See protsess jätkub seni, kuni on uuritud kõik olekud DFA-s. Meetod on joonisel 10.

```
private void ComputeShortestPathsFromInitState()
{
    Queue<int> queue = new Queue<int>();
    queue.Enqueue(initialState);

    while (queue.Count > 0)
    {
        int curState = queue.Dequeue();

        foreach (var nextState in _deltaDictionary[curState])
        {
            if(!_shortestPathsFromInitialState.ContainsKey(nextState.Key))
            {
                _shortestPathsFromInitialState[nextState.Key] = new
             List<int>(_shortestPathsFromInitialState[curState]);

                    _shortestPathsFromInitialState[nextState.Key].Add(nextS
             tate.Key);
                queue.Enqueue(nextState.Key);
            }
        }
    }
}
```

Joonis 10: Meetod lühimate teekondade leidmiseks algolekust kuni iga olekuni.

Meetod *ComputeShortestPathsToFinal* toimib järgmiselt: see itereerib üle kõikide lõppolekute, luues iga lõppoleku jaoks järjekorra andmestruktuuri, et teostada laiuti otsinguga (BFS) läbimist (joonisel 11). Kui järjekord pole tühi, eemaldab see oleku ja kontrollib oma naabreid. Kui naabril ei ole salvestatud lühimat teed või tal on pikem tee, värskendab see selle naabri lühimat teed (joonisel 12) ja paneb selle järjekorda. See

protsess jätkub seni, kuni on uuritud igast lõppolekust kõik kättesaadavad olekud. Meetod värskendab *shortestPathsToFinalStates* muutujat, et salvestada lühimad teed igast olekust igasse lõppolekusse.

```
private void PerformBFS(int finalState)
{
    HashSet<int> visited = new HashSet<int>();
    Queue<int> queue = new Queue<int>();
    queue.Enqueue(finalState);

    while (queue.Count > 0)
    {
        int curState = queue.Dequeue();

        foreach (var neighborState in _deltaDictionary.Keys)
        {
            if (_deltaDictionary[neighborState].ContainsKey(curState) &&
            !visited.Contains(neighborState))
            {
                visited.Add(neighborState);
                UpdateShortestPath(neighborState, curState, finalState);
                queue.Enqueue(neighborState);
            }
        }
    }
}
```

Joonis 11: Meetod iga oleku lühima teekonna leidmiseks iga lõppolekuni kasutades graafi BFS algoritmi.

```
private void UpdateShortestPathDictionary(int state, int curState, int
finalState)
{
    if (!_shortestPathsToFinalStates.ContainsKey(state))
    {
        _shortestPathsToFinalStates[state] = new Dictionary<int,
List<int>>();
    }

    if (curState == finalState &&
!_shortestPathsToFinalStates[curState].ContainsKey(finalState))
    {
        _shortestPathsToFinalStates[curState]
.Add(finalState, new List<int>() { curState });
    }

    List<int> newPath = new List<int> { state };
    InitializeShortestPathsForState(state, finalState);
    InitializeShortestPathsForState(curState, finalState);
    newPath.AddRange(_shortestPathsToFinalStates[curState][finalState]);
    if (newPath.Last() != finalState)
    {
        newPath.Add(finalState);
    }
    _shortestPathsToFinalStates[state][finalState] = newPath;
}
```

Joonis 12: Meetod lühima teekonna salvestamiseks igast olekust iga lõppolekuni.

GetAllEdgePairs() meetod otsib kõik võimalikud servade paarid DFA-s. See itereerib üle kõigi olekute. Iga oleku puhul itereerib see üle tema naaber olekute ja seejärel omakorda nende naaberolekute naaber olekute. Iga kolme oleku (allikas, sihtmärk1, sihtmärk2) kombinatsiooni jaoks moodustab see servade paari ja lisab selle unikaalsuse tagamiseks *HashSeti*. Meetod on joonisel 13.

```
public List<(int startState, int middleState, int endState)>
GetAllEdgePairs()
{
    HashSet<(int, int, int)> edgePairs = new HashSet<(int, int, int)>();

    foreach (var sourceState in delta.Keys)
    {
        foreach (var targetState1 in delta[sourceState])
        {
            foreach (var targetState2 in delta[targetState1.TargetState])
            {
                edgePairs.Add((sourceState, targetState1.TargetState,
targetState2.TargetState));
            }
        }
    }

    return edgePairs.ToList();
}
```

Joonis 13: Meetod kõikvõimalike servade paaride genereerimiseks.

GeneratePaths meetod genereerib teekonnad, mis põhinevad otsitud servade paaridel. Kõiki servade paare läbikäimiseks valib see saadaolevate servapaaride hulgast juhuslikult servapaari. See eraldab servapaarist algus-, kesk- ja lõppoleku. Iga lühima tee jaoks DFA algusest lõppolekusse leiab see lühima tee DFA algolekust kuni servade paari algolekuni (allikas) ja lühima tee servade paari lõppolekust (sihtmärk2) DFA ühte lõppolekusse. Erandina kui algolekust on otsetee ehk üks serv lõppolekusse, siis lisatakse see teekond samuti. Kui soovitakse leida peaaegu vastavat stringi, siis boolean parameetri alusel teostab see loodud teekonnal muudatusi DoAlmostMatching meetodi abil. DoAlmostMatching meetod põhimõtteliselt eemaldab teekonnast seni kaua olekuid kuni viimane olek ei ole lõppolek ehk genereeritav tekst ei vasta antud regulaaravaldisele. Lõpuks salvestab see loodud teekonna muutujasse kui see on kordumatu ning eemaldab antud servade paari *edgePairPool*'ist. Kui kõik servapaarid on läbi käidud, tagastab see loodud teekondade listi. GeneratePaths meetod on joonisel 14.

```
public List<List<(int, int)>> GeneratePaths(bool almostMatch)
{
    var edgePairPool = GetAllEdgePairs();

    List<List<int>> paths = new List<List<int>>();

    Random rand = new Random();

    FindInitStateToFinalStatePairs(almostMatch, paths);

    while (edgePairPool.Count > 0)
    {
        (int, int, int) edgePair =
edgePairPool[rand.Next(edgePairPool.Count)];

        int start = edgePair.Item1;
        int middle = edgePair.Item2;
        int end = edgePair.Item3;

        foreach (var finalState in _shortestPathsToFinalStates[end].Keys)
        {
            var path = new List<int>();
            path.AddRange(_shortestPathsFromInitialState[start]);
            path.Add(middle);
            path.AddRange(_shortestPathsToFinalStates[end][finalState]);

            if (almostMatch)
            {
                DoAlmostMatching(path);
            }

            if (!paths.Any(p => p.SequenceEqual(path)))
            {
                paths.Add(path);
            }
        }

        edgePairPool.Remove(edgePair);
    }

    var allPaths = GetPathsAsStatePairs(paths);

    return allPaths;
}
```

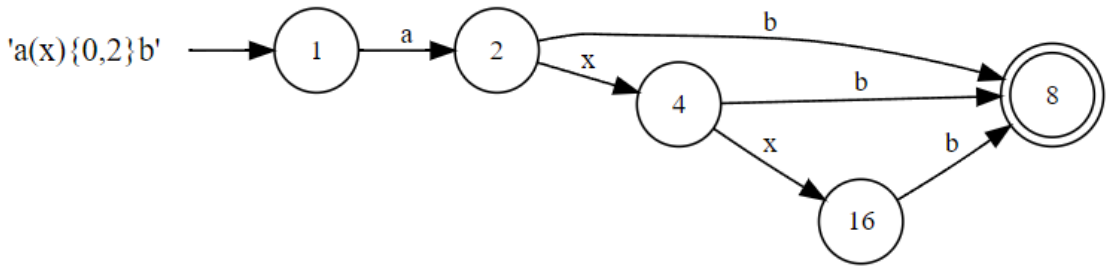
Joonis 14: Meetod teekondade salvestamiseks iga servade paari jaoks.

3.6 Realiseeritud algoritmi funktsionaalsuste ülevaade

Algoritmi implementeerimise tulemusena on võimalik genereerida erinevaid tekste vajalikke parameetritega. Toon mõned näited kasutades implementeeritud algoritmi:

1. a(x){0,2}b

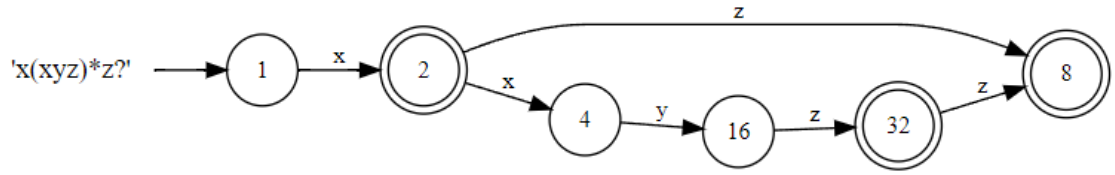
Väljund: ab, axb, axxb.



Joonis 15: Regulaaravaldise „a(x){0,2}b“ automaat graafiliselt.

2. x(xyz)*z?

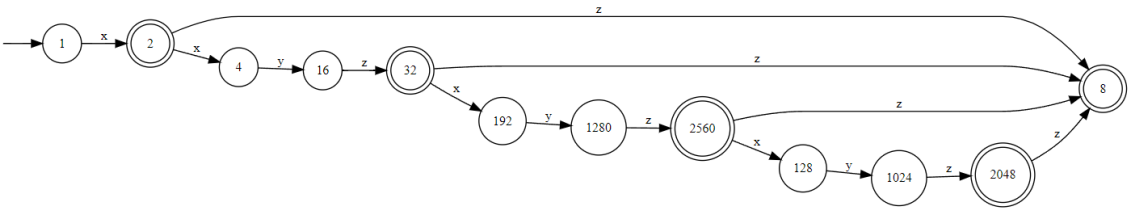
Vaikimisi sätetega automaat ('*' asendatakse vahemikuga {0,1}):



Joonis 16: Regulaaravaldise „x(xyz)*z?“ automaat graafiliselt.

Väljund: x, xxyzz, xxyz, xz.

Sätestades parameetri *starMaxRepeat* = 3, saame automaadi:

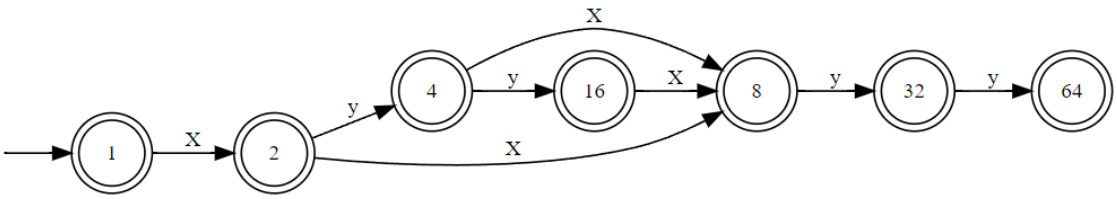


Joonis 17: Regulaaravaldise „x(xyz){0,3}z?“ automaat graafiliselt.

Väljund: x, xxyzxyz, xxyzxyzxyz, xxyzz, xxyz, xxyzxyzxyzz, xxyzxyzz, xz.

3. (X(y)*)*

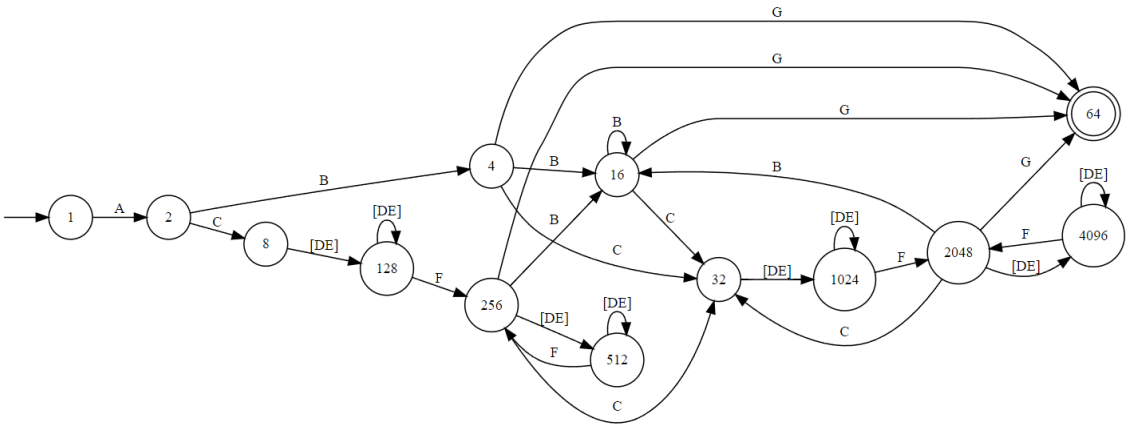
Sätestades parameetri starMaxRepeat = 2 ('*' asendatakse vahemikuga {0,2}), saame automaadi „(X(y){0,2}){0,2}“:



Joonis 18: Regulaaravaldise „(X(y)*)*“ automaat graafiliselt.

Väljund: X, Xy, XyX, XyXy, XyXyy, Xyy, XXyy, XyyX, XyyXy, XyyXyy, XXy, XX.

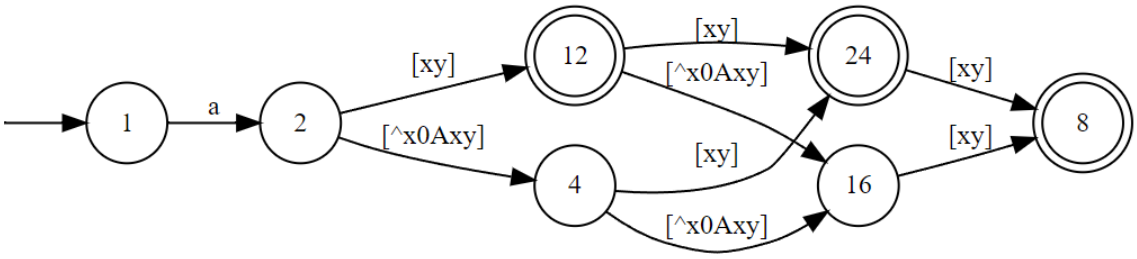
4. A(?:B|C(?:(?:D|E)+F)+)+G



Joonis 19: Regulaaravaldise „A(?:B|C(?:(?:D|E)+F)+)+G“ automaat graafiliselt.

Tekste on kokku 32. Kui anda parameetrina ette, et teksti pikkus peab olema vahemikus 4-5, siis saame väljundiks: ACEFG, ABBBG, ABBG.

5. a.*(x|y)



Joonis 20: Regulaaravaldise „a.*(x|y)“ automaat graafiliselt.

Üks võimalikest väljundite variantidest: agpy, aBxx, a\y, ayyy, ay7y, axx, ax.

6. Tekste saab ka genereerida keerukamate regulaaravaldiste põhjal (võrdlus tabelis 4):

    1. ((\w|\d|\-|\.)+)@{1}(((\w|\d|\-){1,67})|((\w|\d|\-)+\.(\w|\d|\-){1,67}))\.(([a-z]|[A-Z]|\d){2,4})(\.([a-z]|[AZ]|\d){2})?)

    2. (([A-Za-z0-9]+ +)|([A-Za-z0-9]+\-+)|([A-Za-z0-9]+\.+)|([A-Za-z0-9]+\++))*[A-Za-z0-9]+@((\w+\-+)|(\w+\.))*\w{1,63}\.[a - zA - Z]{2,6}

    3. ((\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*)\s*[,]{0,1}\s*)+

	Näide 6.1	Näide 6.2	Näide 6.3
Tekstide arv	605	2575	4046
Kulunud aeg (sekundites)	26.10	71.28	26.78
Automaadis sõlmesid	153	88	19
Lõppolekute arv	8	10	7
Miinimum pikkus	6	5	5
Maksimum pikkus	77	72	10
Mediaan pikkus	37	23	8
Keskmine pikkus	38.13	29.86	7.81

Tabel 4: Regulaaravaldiste pealt tekstide genereerimise võrdlus.

3.7 Algoritmi optimeerimine

Optimeerimine on kriitilise tähtsusega selle lahenduse juures. Suuremate ja keerukamate regulaaravaldistega ei pruugi algoritm oma tööd mõistliku aja jooksul lõpetada. Mida keerukam on regulaaravaldis ehk mida rohkem on automaadis olekuid, servi ja lõppolekuid, seda kauem võtab algoritmi jooksutamine. Optimeerimise eesmärgil teisendasin MS Automata teegi Automaton klassi muutuja *delta*, kus hoiustatakse automaadi olekuid ja käike, teistsugusesse andmestruktuuri, et küsida otse kahe oleku kui võtmete järgi nendevahelisi servi. Järgnevalt toon näite, kuidas suutis algoritm genereerida keerukamaid regulaaravaldisi.

1. Kuupäeva regulaaravaldis, mis on pikkusega 6349. Vaata lisa 2.1. [21]

2. URL regulaaravaldis, mis on pikkusega 10 336. Vaata lisa 2.2. [21]

	Kuupäeva regulaaravaldis	URL regulaaravaldis
Genereeritud tekste	7331	791 996
Aeg minutites	01:29.4298845	05:36.2283256
olekuid	1532	4063
Servade paare	8106	19466
Lõppolekuid	150	276
Miinimum pikkus	1	4
Maksimum pikkus	27	79
Mediaan pikkus	8	31
Keskmine pikkus	9.146092	33.67787

Tabel 5: Keerukate regulaaravaldiste pealt tekstide genereerimise võrdlus.

3.8 Tulemuste valideerimine

Tulemuste valideerimiseks testisin nii manuaalselt kui ühiktestidega. Ühiktestidega testisin väljundi valiidsust, st kontrollisin Rex olemasoleva meetodiga *IsMatch* kas kõik regulaaravaldise põhjal genereeritud tekstid vastavad sellele. Lisaks testisin, kas väljundi suurus on õige ehk kas tagastatakse õige arv tekste. Testidega katsin ära ka lõpmatute sümbolite kasutamise, muutes maksimaalset ja minimaalset piirangut nende korduste sätestamiseks. Andes parameetrina minimaalse ja maksimaalse lubatud pikkuse, testisin kas meetod tagastab õige arvu tekste. Lõpuks testisin ka peaaegu vastavust, valideerisin kasutades *IsMatch* meetodit, et kas kõik tagastatavad tekstid ei vasta antud regulaaravaldistele.

4 Tulemuste analüüs

Alljärgnevas peatükis käsitlen käesoleva bakalaureusetöö tulemuste analüüsi erinevaid aspekte, sealhulgas kasutatud tööriistasid, tööprotsessi, lahenduse vastavust püstitatud nõuetele ning testimise analüüsi ja edasiarenduse ettepanekuid.

4.1 Tööriistade analüüs

.NET raamistikuga olin ma varasemalt tuttav. Seega oli seda lihtne ja mugav kasutada. MS Automata sisaldas kõike vajalikku oma lahenduse realiseerimiseks. See võimaldab regulaaravaldisi sümbolautomaatideks teisendada, determineerida ja graafiliselt esitada. Sain keskenduda oma algoritmi implementeerimisele. Uue teegiga tutvumine võttis omajagu aega. Seda ei olnud väga lihtne teha, sest automaatide teooria oli samuti minu jaoks uus. Veel kasutasin oma töös automaadi visualiseerimise tööriista [11], mis aitas Automata genereeritud automaatidest pildi ette saada, selle abiga suutsin oma algoritmi välja töötada.

4.2 Tööprotsessi analüüs

Antud bakalaureusetöö keskendus rohkem praktilisele poolele. Kõik algas aurvutiteaduse fundamentaalse teooria nagu selleks on automaatide teooria kohta lugemisest ja selle arusaamisest. Automaatide teooria oli minu jaoks uus teema. Nagu iga uue asjaga, läheb selle mõistmiseks aega. Kuid teooria ise on oma olemuselt lihtne ja kergesti mõistetav. Teise etapina hakkasin uurima MS Automata teeki ja sealseid võimalusi ning piiranguid. Kaardistasin mida teek toetab, mida mitte. Selle jaoks uurisin ka MS Automata teste. Uurimise tulemusena ja töö käigus tulin mõttele lahendada teksti genereerimise probleem just sellise lähenemisega nagu sai implementeeritud. Skoopi alguses ei olnud, kuna ei olnud selge mida ja palju jõuab töö raames teha. Sellele probleemile lahenduse leidmine toimus töö käigus, mis oli ka kõige raskem aspekt tööprotsessi juures.

4.3 Lahenduse vastavus nõuetele

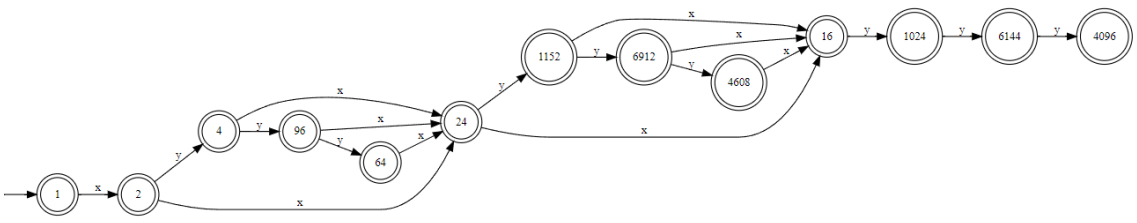
Kõik püstitatud nõuded said täidetud. Lahendus genereerib erinevad tekstid etteantud regulaaravaldiste põhjal kattes ära kõik servade paarid. Kasutaja saab otsustada tagastatavate tekstide minimaalse ja maksimaalse pikkuse. Samuti saab otsustada, kui

palju on kordusi lõppematutel sümbolitel ning valida tagastavateks tekstideks mittevastavad tekstid, mis peaaegu vastavad etteantud regulaaravaldistele.

Lõpmatute sümbolite korduste jaoks otsustasin asendada need vahemikkudega, et leida kõik teekonnad. Selleks pidin aga SFA teisendama DFA-ks, et leida kõik teekonnad graafis, sest algoritmi idee töötab ainult DFA põhjal. Paraku tekitab see jõudluse probleeme, kui DFA on piisavalt suur, siis võtab see liiga kaua aega. Vastavalt selle kumb on tekstide genereerimisel olulisem, tuleks valida, kas kasutada NFA-d või DFA-d. Algoritm töötab mõlema automaadi peal väikeste erisustega. Näiteks peaaegu vaste leidmisel, tuleb mitte ainult eemaldada viimast elementi niikaua, kuni ei olda lõppolekus, vaid tuleb arvestada ka epsilon servadega. NFA puhul ei leia algoritm kõiki teekondi automaadis, sest teekonna valikul leidub ka lühemaid teid tänu epsilon servadele, ning niimoodi ei saa valida kõige pikemat teekonda.

Lühimate teede ja servade paaride genereerimise algoritm on hea lahendus sellele, kuidas katta ära erinevad tekstid, mida on võimalik regulaaravaldise põhjal genereerida. Olemasolev lahendus on terviklik ja töötab korrektselt nii nagu on kirjeldatud. Saab olla kindel, et 100% olekutest ja servade paaridest on kaetud.

Algoritmi rakendamisel tuleb arvestada ühe piiranguga. Käsitsi testides ja tulemusi valideerides, leidsin üles ühe piirjuhu, mis oli kahe silma vahele jäänud. Nimelt kui tekib automaat, kus on palju lõppolekuid, valides servade paari, leitakse kõik teekonnad sellest servade paarist kõikide lõppolekuteni aga teekond algolekust kuni selle servade paarini on lühim võimalik ehk võib tekkida olukord, kus algoritm ei leia kõiki teekondi, sest lõppolek oli varasemalt olemas aga leidus lühem teekond. Paremaks mõistmiseks toon näite, regulaaravaldise “(x(y)*)*” korral loob Automata järgmise graafi, mis on järgmisel joonisel.



Joonis 21: Automaat ainult lõppolekutega.

Algoritm ei moodusta kunagi näiteks teksti varianti “xyxyyx”, mis tegelikkuses sobiks antud regulaaravaldise vasteks. Maha ei salvestata teekonda olekute 1-2-4-24-1152-6912-16 vahel, sest teekond, mis kataks servade paari 2-4-24 ja leiaks kõik teekonnad lõppolekutesse, ei vali teekonda 24-1152-6912-16, sest olekust 24 leidub lühem teekond lõppolekusse 16 kui teekond 24-1152-6912-16. Selle servade paari 2-4-24 valikul kui luuakse teekond, arvutatakse teekonnad lõppolekutesse nii, et alustatakse olekust 24 ja itereeritakse üle lõppolekute ning leitakse lühim teekond nende vahel, ehk olek 24 ja lõppolek 16 vahel on lühim tee 24-16, mitte 24-1152-6912-16. Selline automaat on erandlik ja see erand, kus algoritm ei genereeri kõiki tekstide variante, tekib ainult siis kui lõpmatu korduste arvuga sümbol asub omakorda lõpmatu arvuga sümbolite grupeeringu sees (need asendatakse vahemikkudega) ning on regulaaravaldise lõpus ehk lõppeb sellega. Seega selline olukord tekib siis, kui regulaaravaldis sisaldab lõpmatu korduste arvuga sümbolit, mis asub omakorda teise lõpmatu kordusega sümbolite grupeeringu sees (näiteks (x(y)*)*) ja see komponent asub regulaaravaldise lõpus. Automaat moodustatakse nii, et lõppolekuteni jõudmisel valitakse lühim võimalik teekond, mistõttu kui kõik olekud on lõppolekud võib jääda leidmata teekondi, mis vastavad regulaaravaldisele, kuid ei ole kõige lühemad.

4.4 Testimise analüüs

Kuigi teste on vähe, katavad need ära minimaalse vajaliku. Samas on hea, et testid on omaduspõhised (genereeritud tekst vastab regulaaravaldisele, genereeritud tekst ei vasta regulaaravaldisele, genereeritud teksti pikkus on õige), et need ei sõltuks konkreetsest regulaaravaldisest.

4.5 Edasiarenduse ettepanekud

Edasiarendamise võimalusi on mitmeid. Selle bakalaureuselõputöö raames sai arendatud hea funktsionaalsus, kuid paraku on sellel ka kitsaskohti.

Olenevalt regulaaravaldisest, võib mõne mittedeterministliku sümbolautomaadi determinismlikuks automaadiks teisendamine olla väga ajakulukas. Samuti võib DFA olla algoritmi jaoks liiga suur ehk omada palju olekute paare ja lõppolekuid. Selle probleemi lahendamiseks tuleks edasi mõelda, kuidas saaks sarnaste lõpptulemusteni jõuda NFA-ga.

Edasiaredusena oleks sobilik ka uurida, kuidas saaks algoritmi edasi optimeerida. Üks asi, mida tasub kindlasti edasi vaadata on '*' ja '+' korduste haldamine. Optimeerimise seisukohalt nende sümbolite asendamine vahemikuga ei pruugi olla kõige optimaalsem lahendus, sest see tekitab suurema ja keerukama DFA.

Veel ühe edasiarendusena võiks lisada uue parameetri selleks, et grupeerimise või tehte "(|)" korral saaks otsustada kumba poolt algoritm eelistab ehk kas eelistada esimest või teist valikut.

Kui peaks tekkima vajadus, saaks parametriseerimist teha veel detailemaks, näiteks iga konkreetse grupi/vahemiku korduste arvu määramine eraldi.

Lõppeesmärk on regulaaravaldiste mootoreid benchmarkida. Kuna mõned teised regulaaravaldiste mootorid toetavad ka muid funktsionaalsusi, mida Automata ei toeta, st hetkel ei ole võimalik nende stsenaariumite testimiseks tekste genereerida, siis võib ka Automatat ise täiendada. Vaata MS Automata piiranguid, seal on võimalusi, kuidas saaks Automatat edasi täiendada.

5 Kokkuvõte

Antud bakalaureusetöö eesmärk oli luua tekstide genereerimise lahendus regulaaravaldiste põhjal, kasutades Microsoft Automata teeki. Selleks kasutati MS Automata teeki ja selle Rex alamprojekti, mis võimaldab regulaaravaldiste teisendamist sümbol automaatideks (SFA) ning nende manipuleerimist. Algoritmi põhieesmärk oli katta vajalikud tekstivariandid vastavalt etteantud piirangutele, nagu teksti pikkus, piiramatu kordusega sümbolid regulaaravaldises, peaaegu vastavad tekstid.

Loodud funktsionaalsus võimaldab luua testimiseks vajalikke tekstide komplekte, võttes arvesse antud regulaaravaldisi ja sätestatud piiranguid. See tagab nii täpsete vastete kui ka peaaegu vastavate tekstide loomise. Lahendus põhineb kõikide servade paaride ja nende lühimate teekondade arvutamisel DFA olekute vahel, et tagada tekstide täieliku katvuse vastavalt regulaaravaldistele. Nõuded on täidetud ja eesmärk on saavutatud. Töös on välja toodud ka edasiarenduse ettepanekud.

Kasutatud kirjandus

[1] J. E. F. Friedl, *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006.

[2] J. C. Davis, C. A. Coghlan, F. Servant, and D. Lee, "The impact of regular expression denial of service (ReDoS) in practice: an empirical study at the ecosystem scale," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 246–256. doi: 10.1145/3236024.3236027.

[3] "Mark Twain corpus." [Online]. Available: http://www.gutenberg.org/files/3200/old/mtent12.zip

[4] "Generate Text from Regex - Regular Expression String Generator - Online - Browserling Web Developer Tools." Accessed: May 14, 2024. [Online]. Available: https://www.browserling.com/tools/text-from-regex

[5] D. Scheller, "wimpyprogrammer/regex-to-strings." May 05, 2024. Accessed: May 14, 2024. [Online]. Available: https://github.com/wimpyprogrammer/regex-to-strings

[6] M. Sipser, *Introduction to the theory of computation*, Third edition, International edition. Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning, 2013.

[7] M. Sipser, *Introduction to the theory of computation*, Third edition, International edition. Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning, 2013.

[8] L. D'Antoni and M. Veanes, "The Power of Symbolic Automata and Transducers," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds., Cham: Springer International Publishing, 2017, pp. 47–67. doi: 10.1007/978-3-319-63387-9_3.

[9] M. Veanes, P. De Halleux, and N. Tillmann, "Rex: Symbolic Regular Expression Explorer," in *2010 Third International Conference on Software Testing, Verification and Validation*, Paris: IEEE, Apr. 2010, pp. 498–507. doi: 10.1109/ICST.2010.15.

[10] D. Fisman, H. Frenkel, and S. Zilles, "Inferring Symbolic Automata," *Log. Methods Comput. Sci.*, vol. Volume 19, Issue 2, Apr. 2023, doi: 10.46298/lmcs-19(2:5)2023.

[11] "app." Accessed: May 12, 2024. [Online]. Available: https://cs.taltech.ee/staff/iavara/regex_graph/

[12] "Edotor — Your Favorite Online Graphviz Editor." Accessed: May 09, 2024. [Online]. Available: https://edotor.net/

[13] M. Kulkin, "maximkulkin/hypothesis-regex." Mar. 12, 2024. Accessed: Apr. 24, 2024. [Online]. Available: https://github.com/maximkulkin/hypothesis-regex

[14] "Automata/src/Automata/Rex at master · AutomataDotNet/Automata," GitHub. Accessed: Apr. 24, 2024. [Online]. Available: https://github.com/AutomataDotNet/Automata/tree/master/src/Automata/Rex

[15] N. Van Hoan and P. N. Hung, “Arext: Automatic Regular Expression Testing Tool Based on Generating Strings With Full Coverage,” in *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, Nov. 2021, pp. 1–6. doi: 10.1109/KSE53942.2021.9648604.
[16] “Welcome to Hypothesis! — Hypothesis 6.102.1 documentation.” Accessed: May 14, 2024. [Online]. Available: https://hypothesis.readthedocs.io/en/latest/
[17] “Rex - Regular Expression Exploration,” Microsoft Research. Accessed: Apr. 24, 2024. [Online]. Available: https://www.microsoft.com/en-us/research/project/rex-regular-expression-exploration/
[18] “Automata,” Microsoft Research. Accessed: Apr. 23, 2024. [Online]. Available: https://www.microsoft.com/en-us/research/project/automata/
[19] S. Owens, J. Reppy, and A. Turon, “Regular-expression derivatives re-examined,” *J. Funct. Program.*, vol. 19, no. 2, pp. 173–190, Mar. 2009, doi: 10.1017/S0956796808007090.
[20] “gedigeorg/Automata at test.” Accessed: May 23, 2024. [Online]. Available: https://github.com/gedigeorg/Automata/tree/test
[21] A. Gallant, “BurntSushi/rebar.” May 17, 2024. Accessed: May 19, 2024. [Online]. Available: https://github.com/BurntSushi/rebar
[22] “app.” Accessed: May 12, 2024. [Online]. Available: https://cs.taltech.ee/staff/iavara/regex_graph/

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Gedi Georg

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tekstide genereerimine regulaaravaldiste põhjal“, mille juhendaja on Bahdan Yanovich ja kaasjuhendaja on Ian Erik Varatalu.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

20.05.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Keerulised regulaaravaldised

1. Kuupäeva regulaaravaldis [21]

2. URL regulaaravaldis [21]