TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Alo Aasmäe

Keele täiendamine lausearvutusvalemite translaatori näitel

Bakalaureusetöö (9 EAP)

Juhendaja: Ahti Peder, PhD

Tartu 2019

Keele täiendamine lausearvutusvalemite translaatori näitel

Lühikokkuvõte:

Mitmeid kombinatoorikaprobleeme saab kirjeldada lausearvutusvalemite abil. Antud loogikavalemi kehtestavate väärtustuste leidmisel on võimalik teada saada püstitatud ülesande võimalikud lahendid, milleks ongi valemi kehtestavate väärtustuste arv.

Sellise lähenemisviisiga ülesannete hõlbustamiseks on loodud lausearvutusvalemite translaator, mis teisendab LaTeXformaadis parametriseeritud sisendvalemi kindla formaadiga parameetriteta lausearvutusvalemiks. Teisendatud valemit saab seejärel kasutada mõne kehtestavate väärtustuste loenduriga, et leida valemi kehtestavate väärtustuste arv. Antud töö eesmärgiks oli anda ülevaade, kuidas teha läbi ühe programmeerimiskeele täiendamine eelmainitud lausearvutusvalemite translaatori näitel.

Töö tulemusena täienes lausearvutusvalemite translaator nii, et translaator on nüüd võimeline töötlema keerulisemaid loogikavalemeid, mis kasutavad näiteks mõne etteantud graafi täiendit. Selle näitamiseks analüüsiti läbi üks Stamm-Wilbrandti poolt kirjeldatud probleem graafi klikkideks jaotamise võimaluse kontrollimise kohta, mida varem translaator ei olnud võimeline läbi töötlema. Täiendatud translaatorprogramm aitab viia läbi selliseid teaduslikke eksperimente, kus on teada mõne probleemi rahuldavate seisundite arv, aga puudub nende seisundite täielik kirjeldus.

Võtmesõnad:

Lausearvutus, kehtestamine, kombinatoorika, programmeerimiskeel, grammatika, graaf

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Improving a Language for a Compiler of Propositional Formulae

Abstract:

Many combinatorial problems can be described by propositional formulae. By finding possible interpretations that satisfy a given formula, it is possible to find the solution to the problem by counting the number of satisfiable interpretations.

There has been created a translator for propositional formulae to tackle combinatorial problems in such a way, which translates a parametrized formula described in LaTeXto a formula without parameters. This new formula can be used to find the number of satisfiable interpretations by using a satisfiability counter. The aim of this thesis is to give an overview of how to modify and improve a programming language using this existing translator as an example.

As a result of the work of this thesis the translator is now able to process more complicated descriptions of logical formulae, for example such formulae that describe a problem using complements of a graph. To show this result, a formula for describing the cliques of a graph by Stamm-Wilbrandt, a problem that was not previously processable by the translator, was analyzed and validated.

The improved translator is useful for conducting scientific experiments, where there is known the amount of satisfiable solutions, but the descriptions of those solutions is not entirely known.

Keywords:

Propositional calculus, satisfiability, combinatorics, programming language, grammar, graph

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Matemaatiline loogika ja arvutiteadus on omavahel tihedalt seotud [1]. Tihtipeale kasutatakse parameetritega lausearvutusvalemeid, et kirjeldada mõnda arvutiteaduslikku probleemi. Probleemile on võimalik leida lahendus, kui loendada loodud lausearvutusvalemi kehtestavad väärtustused. Just nende väärtustuste kaudu saab uurida, milliste juhtude korral on kirjeldatud olukord võimalik.

Seega, kui on teada, mitu võimalikku lahendit võib ühel ülesandel olla, aga puudub vastav valem nende lahendite väljendamiseks, siis on võimalik kirjeldada probleemi lausearvutusvalemina ning uurida selle kehtestavate väärtustuste arvu. Juhul kui loodud valemi kehtestavate väärtustuste arv on võrdne võimalike lahendite arvuga, siis see võib viidata sellele, et valem kirjeldab püstitatud probleemi korrektselt.

Sellise lähenemisega ülesannete lahendamiseks on loodud programm, millel on kaks põhiosa: kompilaator ja selle eelprotsessor [2]. Programm teisendab formaadis TEXkirjeldatud parameetritega lausearvutusvalemi parameetriteta valemiks, kasutades just selleks otstarbeks loodud programmeerimiskeelt [3]. Teisendus väljastatakse organisatsiooni DIMACS (the Centre for Discrete Mathematics and Theoretical Computer Science) poolt loodud lausearvutusvalemite vormingus *satex* [4]. Saadud väljundit saab omakorda sisestada eraldiseisvasse programmi, mis leiab parameetriteta loogikavalemi korral kõik selle kehtestatavad väärtustused.

Käesoleva töö eesmärgiks on anda ülevaade, kuidas teha läbi ühe programmeerimiskeele täiendamine eelmainitud lausearvutusvalemite kompilaatori näitel. Selle jaoks täiendatakse programmeerimiskeele *grammatikat* ehk reegleid, mille põhjal kompilaator teisendab sisendteksti väljundkujule. Lisaks uuritakse kompilaatori eelprotsessorit, mis tegeleb kompilaatori sisendi eeltöötlusega. Grammatika juures piirdutakse graafi täiendi võtmise kirjelduse töötlemisega. Näiteks kompilaator aktsepteerib sellise lausearvutusvalemi kirjeldust, mis käsitleb mõne serva sisaldumist graafis, aga serva mittesisaldumisega kirjeldusi ei aktsepteerita. Eelprotsessorile lisatakse juurde võimalus esitada ühtselt

korduvaid lausearvutusvalemi osasid märksõnaga `\forall`, et suurendada kasutajamugavust.

Täiendamise tulemusena saab anda programmile sisendiks keerulisemaid TEXkujul valemeid üldistatumal kujul ning kontrollida saadud väljundi kaudu selle kirjelduse korrektsust. Näiteks, graafi täiendi kirjeldamise võimaldamine annab võimaluse kontrollida graafi klikkide, teatud tingimustega alamgraafide olemasolu. Klikkidele on leitud erinevaid rakendusi mitmes valdkonnas, alates bioinformaatikast kuni sotsiaalteadusteni [5, 6]. Seega, täiendatud valmisprogramm on abiks teaduslike eksperimentide läbiviimisel, kus on teada mõne probleemi rahuldavate seisundite arv, aga puudub nende seisundite täielik kirjeldus.

Antud töö sisu on jaotatud viieks peatükiks. Esimeses peatükis antakse ülevaade käsitletavast kompilaatorist ja selle eelprotsessorist ning nende täiendamise vajadusest. Teises peatükis tuuakse näide programmi rakendamisest. Selles esitatakse kirjeldus lippude paigutamisest malelaual ning kirjeldatakse, kuidas kompilaatorit saab kasutada selle probleemi kirjelduse korrektsuse kontrollimiseks. Kolmas ja neljas peatükk käsitlevad vastavalt eelprotsessori ja kompilaatori täiendamist. Viiendas peatükis tuuakse näide graafi klikkide kontrollimise kirjeldusest, mida kompilaator varem ei olnud võimeline töötlema [7].

1 Ülevaade loogikavalemite teisendusprogrammist

Järgnevalt antakse ülevaade programmiga seonduvatest mõistetest, translaatori ja selle eelprotsessori tööpõhimõtetest ning nende täiendamise vajadusest.

1.1 Seonduvad mõisted

Käesoleva programmi raames on lausearvutusvalemite esitamiseks tutvustatud kahte mõistet: metamuutuja ning lausearvutusvalemite pere [3]. *Metamuutujaks* nimetatakse loogikavalemis sellist muutujat, mille kaudu saab paljundada mõnda loogikavalemi alamvalemit nii mitu korda, kui palju metamuutuja muutumispiirkond seda võimaldab. Teisisõnu on tegemist programmeerimise mõttes tsüklimuutujaga, kus iga iteratsiooni korral omandab metamuutuja muutumispiirkonnas järgmise võimaliku väärtustuse. Näiteks, olgu meil valem 1, kus i on metamuutuja.

$$\bigwedge_{1 \leq i \leq 3} (x_i) \tag{1}$$

Antud valem on võrdväärne loogikavalemiga 2, mille kehtestavate väärtustuste arv on 1, kuna antud valem on tõene parajasti siis, kui kõik kolm x_i väärtustust on tõesed.

$$x_1 \wedge x_2 \wedge x_3 \tag{2}$$

Lausearvutusvalemite pereks nimetatakse sellist loogikavalemit, mis sisaldab nii määratud väärtustega metamuutujaid kui ka kasutaja poolt väärtustatavaid parameetreid, millele omistatakse väärtus programmi käitamisel [3].

1.2 Eelprotsessor

Eelprotsessoriks nimetatakse riist- või tarkvara, mis sooritab mõne programmi sisendile ettevalmistavat töötlust [8]. Lausearvutusvalemite kompilaatori käitamisel töödeldakse

sisend esmalt läbi eelprotsessori. Selle käigus tehakse vajalikud asendused vastavalt asendamisreeglitele ning valemite parametriseerimine ehk muutujatena defineeritud osade lahti kirjutamine [2]. Eelprotsessori teisenduste tõttu on võimalik anda programmile ette selline sisend, mis ei pea tingimata vastama täielikult eeldefineeritud grammatikareeglitele. Eelprotsessori lubatud mööndused tõttu teisendatakse sisult sarnane sisend kompilaatorile üheselt aktsepteeritavale kujule. Teisisõnu on võimalik eelprotsesori kaudu laiendada keelele vastuvõetavat sisendit.

1.3 Translaator

Järgnev lõik tugineb A. Isotamme raamatule translaatorite koostamisest [9]. *Translaator* on programm, mis võtab sisendiks mõnes keeles kirjutatud programmi ning tagab selle sisu täitmise. Translaatoreid saab eristada kompilaatoriteks või interpretaatoriteks, kuid esineb ka programme, mis saavad töötada mõlemas režiimis. Kompilaatori töö tulemusena tõlgitakse programmi sisend ühest programmeerimiskeelest teise, kuid interpretaator töötab lähteprogrammi keele raames. Käesoleva loogikavalemite translaatori korral on tegemist kompilaatoriga, kuna sisendiks on tekst LaTeXformaadis, aga väljund on kahes vormingus, DIMACS-i $satex$ ja tav [3, 4]. Väljundformaat tav on loodud sellise programmi kasutamise otstarbeks, mida saab kasutada A. Isotamme koostatud kehtestavate väärtustuste loenduri sisendina.

1.4 Programmeerimiskeele täiendamine

Olemas on kaks peamist viisi, mille kaudu saab mõnda keelt täiendada. Esimene võimalus on suurendada grammatikareeglite hulka ehk defineerida juurde reegleid, mis laiendavad vastuvõetavat keele sisendit. Grammatikareeglite täiendamine tähendab üldjuhul keelt käsitleva translaatori modifitseerimist, et translaator oskaks sisendit käsitleda ka uue reeglite hulgaga. Teine võimalus keele täiendamiseks on teha seda läbi translaatori

eelprotsessori. Kõiki keele võimalusi ei ole mõistlik viia sisse grammatika tasemel, kuna iga väiksema reegli lisamine, näiteks mõne sõne lubamine nii suure- kui ka väikese algustähega, eeldaks eraldi täiendust translaatorile. Selle asemel on lihtsam teisendada sarnase sisu ja tähendusega sisend eelprotsessori kaudu üheselt vastuvõetavale kujule, mis on olemasoleva grammatika poolt juba lubatud ning mille jaoks ei pea translaatorit eraldi täiendama.

Töö kirjutamise käigus leiti, et ühe õppeotstarbelise keele täiendamist sellisel kujul, muutes nii translaatorit kui ka selle eelprotsessorit, ei ole varem teadaolevalt tehtud. Tegemist on kindla eesmärgi täitmiseks loodud süsteemi ja keelega, mistõttu vajadus keele täitmiseks tuleneb päriselulisest puudusest ning vajab lähtekoodis spetsiifilisi muudatusi. Selleks, et säilitada kompilaatori lähtekoodi algkuju, Üldotstarbeliste programmeerimiskeelte korral üldjuhul samamoodi ei läheneta.

Küll aga on uuritud, kuidas realiseerida objektorienteeritud keelte, näiteks Java, puhul laiendatud kompilaatorite arhitektuuri (ingl. *extensible compilers*) [10]. Laiendatud kompilaatoritele saab esitada täiendusi lähtekoodi taaskasutatavate komponentidena, kus lähtekoodi ennast ei muudeta. Sedasi saab säilitada lähtekoodi varasemaid versioone ning realiseerida täiendusi, mis on teineteisest sõltumatud. Antud töö raames otsustati sellist lähenemist mitte kasutada, kuna realiseeritavad täiendused piirduvad vaid paari reegli lisamisega, millele ei ole näha tulevikus lisareeglite lisamise vajadust. Lisaks, kuna kompilaator ei ole algselt sellisel arhitektuuril üles ehitatud, siis see nõuaks terve programmi suuremahulist ümberkirjutamist [9].

1.5 Eelprotsessori täiendamise vajadus

Hetkel on võimalik operaatorite `\bigwedge` ($\bigwedge$), `\bigwee` ($\bigvee$), `\bigoplus` ($\bigoplus$) abil tekitada lausearvutusvalemis tsükleid metamuutujate abil. See tähendab, et näiteks n metamuutuja, mille alumine ja ülemine rada on vastavalt 1 kuni 5, konjunktsiooni kujutamiseks tuleks lausearvutusvalem panna kirja valemina 3, kus sulgude sees olev

sisu on ülejäänud lausearvutusvalemi osa, mida hakatakse tsüklina läbi käima 5^n korda.

$$\bigwedge_{1\leq a_1\leq 5}\bigwedge_{1\leq a_2\leq 5}\ldots\bigwedge_{1\leq a_n\leq 5}(...) \tag{3}$$

Iga operaator võimaldab parajasti ainult ühe metamuutuja kasutamist tsükli koostamiseks, kuid metamuutujaid võib operaatoris veel esineda, näiteks operaatori raja järel lisatingimustes. Valemi 3 kirja panemine nõuab kasutaja poolt eelkõige olemasoleva valemi osa korduvat esitamist, mis raskendab valemi loetavust. Vastuvõetava sisendi mugavdamiseks lisatakse eelprotsessori ülesannete hulka oma sisult rekursiivse operaatori `\forall` ehk $\forall$ töötlemine. See tähendab, et translaatorile saaks anda sisendiks valemi 4,

$$\forall(a_1, a_2, ..., a_n \leq 5)\bigwedge(...) \tag{4}$$

mis teisendatakse samale laiendatud kujule kui valem 3. Sellise teisenduse võimaldamine annab kasutajale võimaluse panna loogikavalemeid kirja arusaadavamal kujul, mille kaudu omakorda saab kirjeldada mõnda kombinatoorikaprobleemi loetavamalt.

Käesolev keel lubab metamuutujate muutumispiirkonna määramisel kasutada operaatorit =, mis ei oma `\forall` korral metamuutujate defineerimisel sisulist tähendust. Kui määrata metamuutuja a muutumispiirkonnaks selline rada, mis ainult märki =, näiteks $a = 5$, siis kaob metamuutujal kui tsüklimuutujal tähendus ära, kuna tegemist oleks siis ainult üht väärtustust omava muutujaga. Lisaks lubab keel kasutada operaatoreid $\geq$ ja >, aga nendega defineeritud radasid saab kirja panna ka operaatoritega $\leq$ ja <. Näiteks, $a > 5$ asemel on sama rada võimalik esitada kui $5 < a$. Töö raames jäetakse välja võimalus täpsustada `\forall` korral operaatorid =, $\leq$ ja <, et tagada parem loetavus ning ühesem arusaamine.

1.6 Translaatori täiendamise vajadus

Translaator on võimeline töötlema ka selliseid loogikavalemeid, mis käsitlevad mõnda graafiprobleemi. Seda on võimalik teha läbi graafi servade ja tippude hulkade. Graafe

on võimalik esitada programmile kasutades tekstifaili, kuhu on kirja pandud tühikutena eraldatud graafi serva otstippude numbrid, kusjuures siinkohal on tehtud eeldus, et graafi tippude nummerdamine algab alati numbrist 1. Näide esitatavast graafifailist on leitav lisast 1 täpsustatud repositooriumist failina *graaf1.txt*.

Hetkel on võimalik esitada lausearvutusvalemites mõne tippudest (b, c) koosneva serva a kuuluvust graafi G servade hulka $E(G)$ kolmel erineval viisil:

1. $a \in E(G)$ ehk serva a kuuluvust hulka $E(G)$;
2. $\{b, c\} \in E(G)$ ehk tippude paari (b, c) kuuluvust hulka $E(G)$;
3. $a \leftarrow \{b, c\} \in E(G)$ ehk nii serva a kui ka tippude paari (b, c) kuuluvust hulka $E(G)$.

Erinevaid esitusviise kasutatakse olenevalt sellest, kui detailselt soovitakse graafi serva esitada, olgu see ainult servana, tippude paarina või mõlema kombinatsioonina. Translaator ei oska töödelda selliseid lausearvutusvalemeid, mis käsitlevad graafi servade hulka mittekuulumist ehk kujul $\notin E(G)$. Muutuja mittekuulumisel hulka on võimalik kirja panna selliseid loogikavalemeid, mis tegelevad graafi servade hulga täiendi võtmisega. Täiendi võtmine on ainuke graafioperatsioon, mida translaator ei oska veel töödelda. Täiendi lisamisel saab lausearvutusvalemitega kirjeldada suuremas mahus graafidega seotud kombinatoorseid probleeme.

Siinkohal tasub välja tuua, et translaator on võimeline töötlema ka selliseid loogikavalemeid, mis käsitlevad mõne tipu kuulumist graafi G tippude hulka $V(G)$. Selle töö raames ei viida sisse täiendust, mis võimaldaks võtta täiendit graafi tippude hulgast, kuna ei leitud sellist olukorda, kus oleks tarvis kasutada graafi tippude täiendit. Tippude hulga täiendi võtmist ja üheselt mõistmist raskendab asjaolu, et etteantud graafiformaat on esitatud sellisel kujul, kus on välja toodud ainult graafi servad ning nendega seonduvad tipud. Sellise formaadiga ei saa esitada graafe, mis sisaldavad näiteks selliseid tippe, millel puudub serv mistahes muu tipuga.

2 Näide programmi rakendamisest

Programmi töökäigu parema ettekujutuse saamiseks on järgnevalt toodud kirjeldus eelprotsessori ja translaatori rakendamisest, kasutades näitena ülesannet lippude paigutamisest malelaual.

2.1 Lippude paigutamine malelauale

Ülesande püstitus on järgmine: paigutada tühjale $n \times n$ malelauale n lippu nii, et ükski lipp ei oleks ühegi teise lipuga tules. Teada on võimalike lahendite arv erinevate malelaua suuruste korral [11]. Kontrollimaks kirjelduse korrektsust, saab valemi anda sisendina translaatorile. Translaatorist saadav väljund on omakorda sisendiks kehtestavate väärtustuste loendajale, mis annab teada, mitu võimalikku tõeväärtust on valemil.

Tabel 1. Mitte tules olevate lippude paigutamine $n \times n$ malelauale.

n	1	2	3	4	5	6	7	8	9	10
Võimaluste arv	1	1	0	0	2	10	40	92	352	724

Sobivate seisundite arvu saab väljendada lausearvutusvalemite pere 5 kehtestavate väärtustuste arvuna, mis koosneb kuuest alamvalemist. Valemite 5 kirjeldus lippude mitte tules olemiseks on järgmine:

1. valemite f_1 ja f_6 järgi võib malelaua igas veerus ja reas olla ülimalt üks lipp;
2. valemite f_2 ja f_3 järgi võib malelaua igal alam- ja peadiagonaalil olla ülimalt üks lipp;
3. valemite f_4 ja f_5 järgi võib malelaua igal kõrvalisel alam- ja peadiagonaalil olla ülimalt üks lipp;

$$f_1 \& f_2 \& f_3 \& f_4 \& f_5 \& f_6, \text{ kus}$$

$$\begin{aligned}
f_1 &\equiv \bigwedge_{1\le i\le n} (exactlyone(x_{i,j} : 1 \le j \le n)),\\
f_2 &\equiv \bigwedge_{1\le k\le n} (atmostone(x_{i,i-k+1} : k \le i \le n)),\\
f_3 &\equiv \bigwedge_{1\le k\le n} (atmostone(x_{i,k+i-1} : 1 \le i \le n-k+1)),\\
f_4 &\equiv \bigwedge_{1\le k\le n} (atmostone(x_{i,n-k-i+2} : 1 \le i \le n-k+1)),\\
f_5 &\equiv \bigwedge_{1\le k\le n} (atmostone(x_{i,n-i+k} : k \le i \le n)),\\
f_6 &\equiv \bigwedge_{1\le i\le n} (exactlyone(x_{j,i} : 1 \le j \le n)).
\end{aligned} \tag{5}$$

Valemite pere 5 osade kompaktsemaks ja arusaadavamaks esitamiseks on kasutatud operaatoreid $exactlyone$ ja $atmostone$, mille laiendatud kujud on esitatud valemitena 6 ja 7 [2].

$$exactlyone(x_i : 1 \le i \le n) \equiv \bigvee_{1\le i\le n} x_i \& \bigwedge_{1\le i,j\le n; i<j} \overline{x}_i \vee \overline{x}_j \tag{6}$$

$$atmostone(x_i : 1 \le i \le n) \equiv \bigwedge_{1\le i,j\le n; i<j} \overline{x}_i \vee \overline{x}_j \tag{7}$$

Mõlemad operaatorid tekitavad metamuutujatest tingitud tsükli, kusjuures metamuutujate muutumispiirkond on defineeritud operaatori alamvalemi järel. Operaatori $exactlyone$ väärtus on tõene siis, kui täpselt üks tema alamvalemitest on tõene ning $atmostone$ väärtus on tõene siis, kui ülimalt üks tema alamvalemitest on tõene.

2.2 Eelprotsessori rakendamine

Ülesande kirjelduse korrektsuse kontrollimiseks tuleb anda programmile käsureal ette vastavat valemit kujutav $.tex$ või $.ltx$ laiendusega fail. Programmile argumendiks

antud sisendfail töödeldakse esmalt läbi eelprotsessori, mis teisendab sisendvalemi programmile üheselt vastuvõetavasse formaati. Teisenduste kaudu eemaldatakse üleliigsed sümbolid, näiteks tühikud ja reavahetused, ning asendatakse LaTeXkeele mõttes võrdväärsete sõnedega, mis on defineeritud grammatika poolt kui vastuvõetavad [2]. Lisaks, kui sisendfail sisaldab lausearvutusvalemite peret ehk antud loogikavalem koosneb mitmest alamvalemist, siis eelprotsessor teisendab valemi sellisele kujule, kus kõik alamvalemid on peavalemis lahti kirjutatud.

```
$$f_{1} \& f_{2} \& f_{3}  \& f_{4} \& f_{5} \& f_{6} $$
$$f_{1}=\bigwedge \limits_{1\leq i\leq n}(Exactlyone(x_{i,j} : 1\leq
   j\leq n))$$
$$f_{2}= \bigwedge \limits_{1\leq k\leq n}(at_most_one(x_{i, i k+1} :
   k\leq i\leq n))$$
$$f_{3}= \bigwedge \limits_{1\leq k\leq n} (atmostone(x_{i, k+i 1} :
   1\leq i\leq n k+1))$$
$$f_{4}= \bigwedge \limits_{1\leq k\leq n} (atmostone(x_{i, n k i+2}
   : 1\leq i\leq n k+1))$$
$$f_{5}= \bigwedge \limits_{1\leq k\leq n} (atmostone(x_{i, n i+k}: k
   \leq i\leq n))$$
$$f_{6}=\bigwedge \limits_{1\leq i\leq n}(exactlyone(x_{j,i} : 1\leq
   j\leq n))$$
```

Joonis 1. Lippude ülesande kirjeldus $.tex$ tekstina.

Valemite pere 5 kujutuse $.tex$ formaadis joonisel 1 ning sellele järgneva eelprotsessori teisenduse joonisel 2 korral on märgata, et valem on teisendatud kompaktsemale kujule. Esmalt on näha, et kõik peavalemi osad $f_1, ..., f_6$ on asendatud alamvalemite sisuga. Eemaldatud või lühendatud on ebavajalikku teksti. Näiteks märksõna `\limits` abil on algvalemis kujutatud konjunktsiooni `\bigwedge` metamuutuja ülemist ja alumist rada, mis on eelprotsessori poolt asendatud lühema, aga samaväärse versiooniga `\bigwedge_`.

```
#(\bigwedge_{1\leqi\leqn}(exactlyone(x_{i,j}:1\leqj\leqn)))\&(\
    bigwedge_{1\leqk\leqn}(atmostone(x_{i,i k+1}:k\leqi\leqn)))\&(\
    bigwedge_{1\leqk\leqn}(atmostone(x_{i,k+i 1}:1\leqi\leqn k+1)))
    \&(\bigwedge_{1\leqk\leqn}(atmostone(x_{i,n k i+2}:1\leqi\leqn k
    +1)))\&(\bigwedge_{1\leqk\leqn}(atmostone(x_{i,n i+k}:k\leqi\leqn)
    ))\&(\bigwedge_{1\leqi\leqn}(exactlyone(x_{j,i}:1\leqj\leqn)))#
```

Joonis 2. Eelprotsessori väljund.

Lisaks on viidud operaatorite nimetused ühesele kujule, kust on eemaldatud alakriipsud ning suured tähed asendatud väikestega.

Eelprotsessori poolt tagastatav valem salvestatakse $.frm$ formaadis faili, mida saab edaspidi anda otse translaatorile nii, et ei ole tarvis juba teisendatud valemit uuesti eelprotsessori kaudu läbi töödelda. Eelprotsessori töö edukal lõpetamisel edastatakse loodud $.frm$ fail automaatselt edasi translaatorile.

2.3 Translaatori rakendamine

Teisendatud valemikuju antakse peale eelprotsessori töötlust sisendiks translaatorile. Translaatori esimene osa on analüsaator. Analüsaator moodustab analüüsipuu, mida saab seejärel translaator kasutada selleks, et läbida lausearvutusvalemi osasid programmiselt. Analüüsipuu loomise käigus logitakse puu loomise protsess $.htm$ laiendiga faili, mille näide on toodud lisas 1 täpsustatud repositooriumis failina $queens.htm$. Tekkinud analüüsipuu formaat on täpsemalt kirjeldatud raamatus "Translaatorite tegemise süsteem"[9].

Antud ülesandes sõltub kehtestavate väärtustuste arv parameetrist *n*, mistõttu peab kasutaja täpsustama käsureal selle väärtust. Translaatori töö tulemusena luuakse $.sat$ fail, mille sisuks on parameetriteta lausearvutusvalem *satex* formaadis [2]. Lisas 1 täpsustatud repositooriumi kaustas $queens$ on näide väljundfailist $queens.tav$, mis tekib siis, kui

käitada programm parameetriga $n = 8$.

Kui anda saadud väljundfail argumendiks spetsiaalsele programmile, mis kontrollib $satex$ formaadis kirjeldatud parameetriteta lausearvutusvalemi kehtestavate väärtustuste arvu, osutub, et võimalikke kehtestavaid väärtustusi sellel valemil on 92. See on vastavuses juba teadaoleva võimaluste arvuga tabelis 1. Kirjelduse korrektsuse õigsusele annab kinnitust asjaolu, et iga muu parameetri n väärtuse korral tuleb sellele vastav õige kehtestavate väärtuste arv. Saadud tulemus annab kindlust selles osas, et esitatud valem tõepoolest kirjeldab püstitatud ülesannet korrektselt.

3 Eelprotsessori täiendamine

Järgnevalt on kirja pandud nõuded, mis peavad etteantud valemil olema täidetud, et makro nimega `\forall` laiendaks valemit etteantud metamuutujate võrra.

3.1 Operaatori `forall` tingimused

Olgu meil lausearvutusvalem, mille mingi alamosa koosneb järgnevatest osadest ja võtmesõnadest:

1. `"\forall"`, mis on märgend makro alustamiseks;
2. `rada1` ja `rada2` metamuutujate väärtuste ülemise ja alumise piiri määramiseks;
3. `op1` ja `op2`, mida kasutatakse metamuutujate väärtuse rajade piiride täpsustamiseks `\leq` või $<$ kaudu;
4. `metamuutujad`, mis on loetelu metamuutujatest;
5. `\bigformula` on üks kolmest operaatorist `\bigwedge`, `\bigwee` või `\bigoplus`, millele lisatakse metamuutuja koos rajadega;
6. `korrutatav` on operaatori `\bigformula` järel olev sisu.

```
\forall(rada1 op1 metamuutujad op2 rada2)\bigformula(korrutatav)
```

Joonis 3. Makro definitsioon.

Eelprotsessori täienduse tulemusena teisendatakse joonisel 3 olev kuju joonisel 4 olevale kujule, kus `x1`, `x2` ... , `xn` on muutujad loetelust `metamuutujad`.

Näiteks, olgu meil lausearvutusvalem, mis on kujutatud joonisel 5. Antud valemi osad oleksid eelmainitud nõuete kohaselt järgmised:

1. `\forall` - valemi algus ;
2. `rada1` ja `rada2` - 2 ja 10;
3. `op1` ja `op2` - operaatorid $<$ ja `\leq`;
4. `metamuutujad` - `{a, b}`;
5. `\bigformula` - `\bigwedge`;
6. `korrutatav` - `(x\_{a} \vee x\_{b})`;

```
\bigformula_{rada1 op1 x1 op2 rada2}(
\bigformula_{rada1 op1 x2 op2 rada2}(
...(
\bigformula_{rada1 op1 xn op2 rada2}(korrutatav))))
```

Joonis 4. Makro rakendamise tulemusena saavutatud kuju.

```
\forall(2 < a, b \leq 10)\bigwedge(x_{a} $\vee$ x_{b})
```

Joonis 5. Näide makrot kasutavast valemist.

```
#\bigwedge_{2<a\leq10}(\bigwedge_{2<b\leq10}(x_{a}$\vee$x_{b}))#
```

Joonis 6. Näide makro töö tulemusest.

Makro rakendamise tulemusena teisendatakse antud valem joonisel 6 olevale kujule, kus on lisaks `\forall` teisendusele rakendatud ka kõik ülejäänud eelprotsessori operatsioonid.

3.2 Eelprotsessori täiendus

Eelprotsessor on kirjutatud keeles C ning asub failis $lahti.c$, mis on kättesaadav versioonihaldussüsteemi git kaudu. Link repositooriumile on toodud välja lisas 1. Samas

repositooriumis on olemas ka kompileeritud versioon eelprotsessorist, failina $lahti.exe$. Autori poolt tehtud koodi täiendused on failist leitavad kommentaaride `//ALO AASMÄE OSA ALGUS//` ja `//ALO AASMÄE OSA LÕPP//` vahel. Suurem osa täiendusest on realiseeritud meetodis `LAHTI`, kus lausearvutusvalem läbitakse sümbolhaaval ning sooritatakse ettenähtud teisendused.

Täienduse algoritm on kirjeldatud joonisel 7. Järgnevalt on toodud algoritmis kasutatud funktsioonide täpsemad kirjeldused.

`char *KONTROLLI_PIIRITLEJA_OLEMASOLU(int i, int variandid_suurus, char variandid[], char* kontrollitav)` - funktsioon kontrollimaks, kas ettantud metamuutujas `kontrollitav` leidub piiritleja. Funktsioonis kontrollitakse, kas metamuutuja sisaldab mõnda massiivis `variandid` leiduvat tähemärgist. Tähemärgiks võib olla kas `"<"` või `"\\"` ehk `"\\leq"` algusosa, kus kahte tagurpidi kaldkriipsu kasutatakse ühe kaldkriipsu esindamiseks. Antud funktsiooni rakendatakse ainult esimesele ja viimasele esinevale metamuutujale, sest ainult nende küljes võib olla piiritleja.

`char *LEIA_ALUMINE_PIIRITLEJA (char* piiritleja_identifikaator, char metamuutuja)` - funktsioon, mis leiab ja tagastab etteantud metamuutujast alumise piiritleja koos vastava raja ehk numbriga. Funktsioonis poolitatakse metamuutuja piiritleja identifikaatori järgi osadeks, kaudu saab kätte esimesest elemendist alumise raja. Tulemusena tagastatakse metamuutujast eraldatud alumise raja number ning piiritleja. Juhul, kui poolitatakse identifikaatori `"\\"` järgi, siis peab tagastusväärtusele `"leq"` ka tagasi lisama, kuna poolitamisel see jäi teise elemendi külge.

`char *LEIA_ULEMINE_PIIRITLEJA (char* piiritleja_identifikaator, char metamuutuja)` - tööpõhimõte sarnaneb alumise piiritleja leidmisele, kuid kuna ülemise piiritleja puhul järgneb piiritleja metamuutujale, siis tuleb esmalt eraldada metamuutuja ning alles siis leitakse piiritleja koos rajaga. Selle jaoks loetakse kõik sümbolid kuni esimese `piiritleja_identifikaator` esinemiseni metamuutuja alla. See tähendab, et metamuutuja ei tohi sisaldada identifikaatorsümbolit, aga piiritleja koos rajadega võib

```
   Sisend: valem - läbitav lausearvutusvalem
1  if "forall" in valem then
2      ASENDA_ESIMENE (valem, "forall","")
3      forall_sisu := "forall"sulgude sees komaga eraldatud elemendid
4      alumine_piir := "1<"
5      ülemine_piir := ""
6      piiritleja_identifikaatorid := ['\', '<']
7      foreach metamuutuja in forall_sisu do
8          if metamuutuja on esimene element then
9              piiritleja_identifikaator =
                 KONTROLLI_PIIRITLEJA_OLEMASOLU (0,
                 len(piiritleja_identifikaatorid), metamuutuja,
                 piiritleja_identifikaatorid)
10             if leidub piiritleja identifikaator then
11                 alumine_piir := LEIA ALUMINE PIIRITLEJA
                     (piiritleja_identifikaator, metamuutuja)
12         else if metamuutuja on viimane element then
13             piiritleja_identifikaator =
                 KONTROLLI_PIIRITLEJA_OLEMASOLU (0,
                 len(piiritleja_identifikaatorid) metamuutuja,
                 piiritleja_identifikaatorid)
14             if leidub piiritleja identifikaator then
15                 ülemine_piir := LEIA_ULEMINE_PIIRITLEJA
                     (piiritleja_identifikaator, metamuutuja)
16     if forall järel ei leidu topeltkaldkriipsu then  korrutatav := "bigwedge"
17     else  korrutatav := forall järel olev operaator
```

```
18
19      korrutatava_sisu = korrutatava järel sulgude sees olev sisu
20      laiendatud = korrutatava_sisu
21      foreach metamuutuja in forall sisu do
22          uus_rada = alumine_piir + metamuutuja + ülemine_piir
23          laiendatud_operaator = korrutatav + "_{"+ uus_rada + "}"
24          laiendatud= laiendatud_operaator + "("+ laiendatud + ")"
25  else
26      Algoritm lõpetab töö
```

Joonis 7. Lausearvutusvalemi operaatorite laiendamine metamuutujatega.

jällegi sisaldada mitu sellist identifikaatorit, kuigi need loetakse siis ühe piiritlejana.

`char *ASENDA_ESIMENE(char *sona, char *mis, char *millega)` - funktsioon, mille kaudu saab asendada muutujas `sona` esimese ettejuhtuva sõne `mis` sõnega `millega`. Esialgses eelprotsessori koodis oli olemas sarnane funktsioon `ASENDA` , mis asendas kõik sõne `mis` esinemised. `ASENDA` funktsioon ei sobinud kasutamiseks algoritmis. Kõigi `\forall` esinemiste asendamise tulemusena ei oleks saanud valemites kasutada rohkem kui ühte `\forall` operaatorit, sest esimene `\forall` töö käigus tehtud teisendu oleks eemaldanud kõik edasised võtmesõnad, millega leida järgmiste samanimeliste makrode algused.

Algoritmi eesmärgiks on kõigepealt eraldada sisendvalemist tähtsaimad osad, mille abil hakata koostama uut, teisendatud valemit. Selle jaoks leitakse kõigepealt `\forall` sulgude seest metamuutujad ning ka nende alumised ja ülemised piirid. On tehtud eeldus, et ülemine piir on alati täpsustatud, aga alumise piiri puudumisel kasutatakse vaikeväärtust 1<.

Peale piiride määramist tehakse kindlaks, mis operaator järgneb `\forall` sisule.

Juhul, kui `\forall` järel pole täpsustatud ühtegi operaatorit, vaid kohe algab sulgude sees ülejäänud valem, siis pannakse laiendatava operaatori vaikeväärtuseks `\bigwedge`. Viimasena sisestatakse iga metamuutuja ette alumine piir ning järele operaator ja eelmine teisendatud metamuutuja. Juhul, kui tegemist on esimese metamuutujaga, sisestatakse eelmise metamuutuja asemel `\forall` järel olnud ülejäänud valemi sisu.

4 Translaatori edasiarendus

Järgnevalt antakse ülevaate, kuidas ühte keelt saab täiendada grammatika muutmise kaudu. Selle jaoks tutvustatakse keele konstruktorit, mille abil saab genereerida keele grammatikaga seonduva abifailide komplekti, mida translaator saab seejärel kasutada sisendi töötlemiseks. Lisaks kirjeldatakse, mis muudatusi peab translaatoris sisse viima, et programm oskaks uut grammatikareeglite komplekti õigesti käsitleda. Viimasena antakse lühiülevaade sellest, mis tõrked tekkisid keele täiendamisel ning kuidas need lahendati.

4.1 Konstruktor

Antud keele grammatika on koostatud keelekonstruktoriga, mis võtab sisendiks keele grammatika kirjelduse failis laiendiga $.grm$ [9]. Konstruktor kontrollib, kas kirjeldatud grammatika on kontekstivaba, pööratav ning kas tegemist on eelnevusgrammatikaga [9]. Nende tingimuste rahuldamisel genereerib konstruktor abifailid, mida analüsaator kasutab omakorda analüüsipuu genereerimiseks. Translaator kasutab analüüsipuud selleks, et teisendada sisendtekst grammatikareeglite poolt defineeritud väljundiks. Näide abifailide komplektist koos grammatikafailiga on toodud välja repositooriumis kokkupakitud failina $forms.zip$.

Abifailidest tähtsaim on semantikafail laiendiga $.sem$. Semantikafaili kaudu saab grammatikareeglid viia kokku analüüsipuu tippudega, andes igale reeglile numbrilise väärtuse. Kuna analüüsipuu läbimine sõltub nendest eeldefineeritud väärtustest, siis keele täiendamisel ehk grammatikareeglite juurde lisamisel peab lähtuma esialgsest semantikafailist. Selle jaoks võimaldab konstruktor uue semantikafaili genereerimise asemel võtta argumendiks olemasoleva reeglite komplekti, millele lisatakse juurde uued nummerdatud grammatikareeglid.

4.2 Grammatikareeglite täiendamine

Keele grammatikas kirja pandud reeglid graafis servade ja tippude hulga sisalduvuse kohta on nähtaval joonisel 8, millega seonduvad abireeglid on kirjeldatud täpsemalt kogu keele grammatikas [2]. Sarnaselt saab esitada graafihulkades mittesisalduvuse reegleid, asendades võtmesõna `\in` mittesisalduvusega `\notin`. Kuna töö raames ei realiseerita graafi täiendi võtmist tippude järgi, siis täiendav reeglikomplekt, mis on kujutatud joonisel 9, põhineb ainult viimasel kolmel reeglil joonisel 8.

```
1 `limit' => `metavariables' \in V(`graphname1' )
2         => `metavariables' \in E(`graphname1' )
3         => `metavariable' \leftarrow `arc' \in E(`graphname1' )
4         => `arc' \in E(`graphname1' )
```

Joonis 8. Graafi sisalduvusega seotud grammatikareeglid.

```
1 `limit' => `metavariables' \notin E(`graphname1' )
2 `limit' => `metavariable' \leftarrow `arc' \notin E(`graphname1' )
3 `limit' => `arc' \notin E(`graphname1' )
```

Joonis 9. Uued, graafi mittesisalduvusega seotud grammatikareeglid.

Kuigi esialgses grammatikafailis on reeglid rühmitatud reegli noolest vasaku poole kaupa, on joonisel 9 kujutatud reeglid lisatud esimestest `` `limit' `` reeglite komplektist eraldi faili lõppu. Sedasi reegleid lisades saab olla kindel, et uue keeleversiooni genereerimisel läbi konstruktori jääb igale eelmisele reeglile esialgne numbriline väärtus. Kui lisada uued reeglid esialgse `` `limit' `` komplekti järele, siis nihkuks iga reegli numbriline väärtus, mille igat muudatust peaks kajastama ka analüsaatori koodis. Grammatikareeglite lõppu lisamisel peab ainult katma need kolm lisandjuhtu, ilma muud koodi muutmata.

4.3 Translaatori täiendus

Translaatori täiendus on kirjutatud keeles C, milles oli kirjutatud ka translaatori esialgne versioon. Translaatori täienduse koodiosa on toodud välja repositooriumis, failis $log32.c$, kommentaaride `//ALO AASMÄE OSA ALGUS//` ja `//ALO AASMÄE OSA LÕPP//` vahel. Joonisel 10 on toodud kirjeldus graafi täiendi leidmise algoritmist, kasutades joonise 9 teisel real kirjeldatud reeglit. Ridadel 1 ja 3 olevate reeglite korral on lähenemine sarnane joonisel 10 olevale algoritmile, kuna need on sisuliselt alamosad reeglist real 2. Analüüsipuu läbimise tööpõhimõtet ning siinkohal seletamata jäetud funktsioonide definitsioone saab täpsemalt lugeda translaatori esialgse realisatsiooni kirjeldusest [2]. Järgnevalt on lahti seletatud algoritmi tööpõhimõte.

Algoritm käivitub siis, kui jõutakse sellise analüüsipuu tipuni, mis käsitleb võtmesõna `\notin`. Esmalt küsitakse kasutajalt graafi G funktsiooniga `ANNA_GRAAF(tipp)`. Selle saab kasutaja anda kui failinimena, milles on kirjeldatud graafi G servad. Sisendfaili põhjal leitakse üles graafi mõõde ehk suurima tipu väärtus. See on vajalik, et luua funktsiooni `LOO_GRAAF_SUURUSEGA(mõõde)` abil selline täisgraaf, mis koosneb tippudest 1 kuni `mõõde`.

Kui täisgraaf on olemas, siis läbitakse esialgse graafi servade hulk, eemaldades iga serva loodud täisgraafist. Tulemusena jääb alles selline graaf, mis on esialgsele graafile täiendgraafiks. Algoritmi viimase osana kasutatakse muutujaid `a_tipp` ja `l_tipp` kui tsüklimuutujatena, et läbida valem täiendgraafi iga serva otstippude kombinatsioonina. Töö lõppedes kustutatakse käsitletavate muutujate hulgast `a_tipp` ja `l_tipp` funktsiooniga `KUSTUTA_MUUTUJATEST(tipp)`, et saaks valemi mõnes muus osas neid uuesti defineerida.

```
1  tipp := analüüsipuu tipp, mis sisaldab märksõnaga notin seostuvat alamosa
2  mõõde := -1
3  graaf = ANNA_GRAAF(tipp)
4  foreach (algtipp, lõpptipp) in graaf do
5      if mõõde < algtipp then  mõõde = algtipp
6      if mõõde < lõpptipp then  mõõde = lõpptipp
7  täisgraaf := LOO_GRAAF_SUURUSEGA(mõõde)
8  foreach (algtipp, lõpptipp) in graaf do
9      foreach (täis_algtipp, täis_lõpptipp) in täisgraaf do
10         if algtipp == täis_algtipp && lõpptipp==täis_lõpptipp then
11             eemalda täisgraafist serv tippudega (täis_algtipp, täis_lõpptipp)
12 a_tipp := läbimata serva esimene otstipp
13 l_tipp := läbimata serva teine otstipp
14 LISA_MUUTUJATESSE(a_tipp)
15 LISA_MUUTUJATESSE(l_tipp)
16 tingimused := tingimused, mis kaasnevad koos valemi tsüklimuutujatega
17 v := ülejäänud valemi tipp, mida saab läbida funktsiooniga LABI_VALEM
18 foreach (algtipp, lõpptipp) in täisgraaf do
19     PANE_UUS_VAARTUS(algtipp, a_tipp)
20     PANE_UUS_VAARTUS(lõpptipp, l_tipp)
21     if KAS_TINGIMUSED(tingimused) then  LABI_VALEM(v)
22 KUSTUTA_MUUTUJATEST(a_tipp)
23 KUSTUTA_MUUTUJATEST(l_tipp)
```

Joonis 10. Lausearvutusvalemi operaatorite laiendamine metamuutujatega.

4.4 Programmi täiendamisel tekkinud tõrked

Täiendi kontrollimise algoritmi koostamiseni eelnes mitu alamprobleemi, mis ei seostunud otseselt arendamise ja koodi kirjutamisega, vaid arenduskeskkonna ning vajalike tööriistade ette valmistamises. Need on sellegipoolest osa programmi täiendamisprotsessist tervikuna. Järgnevalt on toodud suuremad lahendamist nõudnud takistused, et kompilaatorile saaks täiendavat koodi juurde kirjutada ning kompilaatoriprogrammi ennast kompileerida ja käivitada.

Esimeseks takistuseks oli translaatori lähtekoodile ligi saamine. Translaatorile tehti viimati pisitäiendusi aastal 2014, mille failide komplekt on toodud lisatud repositooriumisse kokkupakitud failina $LOG\_COMP\_2014.zip$. Failide komplektis oli olemas küll translaatori kompileeritud variant $log32.exe$, kuid puudus lähtekood $log32.c$. Kuna toona ei kasutatud arendamisel versioonihaldussüsteemi, siis pidi arenduse aluseks võtma mõnest vanemast versioonist pärineva lähtekoodi. Seega, lähim sobiv lähtekood pärineski esialgsest versioonist, mis koostati aastal 2001.

Nüüd, kui lähtekood oli olemas, osutus järgmiseks takistuseks lähtekoodi kompileerimine. Esialgset programmi arendati eelkõige 32-bitise süsteemi jaoks 32-bitise C kompilaatoriga. Tänapäevase, 64-bitise C kompilaatoriga arendamisel osutus, et programm lõpetab suvalistel hetkedel töötamise ja seda arvatavasti mäluhalduse tõttu. Edasiste probleemide vähendamiseks otsustati kasutada vanemat 32-bitist C kompilaatori versiooni, mida kasutati ka 2014. aastal programmi arendamisel. C kompilaator on repositooriumis välja toodud kaustas MinGW.

Viimaseks suuremaks takistuseks oli sobiva ja töötava arenduskeskkonna valimine. Esimese valikuna otsustati kasutada ühte kaasaegsemat arenduskeskkonda CLion [12]. CLion lihtsustas täienduskoodi kirjutamist eelkõige tänu teksti automaatse lõpetamisele ning koodistiili märkuste välja toomisele. Kui aga toimus vahetus 64-bitiselt C kompilaatorilt 32-bitisele, osutus, et CLion võimaldab töötamist ainult 64-bitiste kompilaatoritega. See tähendas, et uue koodi kompileerimine sai toimuda ainult keskkonnaväliselt, mis

märgatavalt aeglustas arendust.

Seetõttu, et jällegi vähendada arenduskeskkonnaga seonduvaid probleeme, otsustati arenduskeskkonnaks võtta kasutusele Dev-C++, mida kasutati ka esialgse translaatori versiooni arendamisel. Dev-C++ ei pakkunud samas mahus funktsionaalust kui keskkond CLion, kuid vähemalt sai programmi kompileerida keskkonnasiseselt ning vajadusel ka siluriga kompileeritud programmi sammhaaval läbi töötada.

Arenduseks ettevalmistamisel tekkinud tõrked ilmnesid suuremas osas seetõttu, et prooviti vanemat süsteemi arendada kaasaegsete vahenditega. Tänapäevased tarkvaravahendid küll hõlbustavad koodi kirjutamist ning aitavad märgata juba kompileerimiseelseid vigu, kuid kasutades neid teadaolevalt töötavaid vahendeid, millega esialgne programm oli loodud, saab kindluse, et arenduse käigus tekib vähem just arendusega seotud vigu.

5 Näide programmi uue versiooni rakendamisest

Selles peatükis tuuakse näide ühest graafiprobleemist, mille kirjeldust ei olnud translaator võimeline töötlema enne `\notin` käsitlemise lisamist. Graafiprobleemi seletamiseks kirjeldatakse esmalt graafidega seotud mõistet *klikk*, tuuakse välja klikkidega seotud probleemi kirjeldus ning näidatakse, kuidas translaatori täiendamise tulemusena saab kontrollida selle kirjelduse korrektsust.

5.1 Graafi klikk

Tänapäeval tuntud kliki mõiste sai alguse D. Luce ja A. Perry artiklist, kus klikiks nimetati rohkem kui kahest inimesest koosnevaid gruppe, kes kõik on omavahel sõbrad [6]. Sellist kliki definitsiooni kasutati selleks, et analüüsida erinevaid sotsiaalsete gruppide elemente ehk selle uurimuse raames anda hinnang inimeste hulga suhetele.

Graafi G *klikiks* K nimetatakse G tippude sellist alamhulka $K \subseteq V(G)$, mille iga kahe erineva tipu $u, v \in K, u \neq v$ jaoks on olemas serv graafis G nende kahe tipu vahel [13]. Teisisõnu, graafi G klikiks nimetatakse G sellist alamgraafi, mis on omaette täisgraaf.

Klikke saab kasutada selleks, et leida ja rühmitada sarnaste omadustega andmed. Tänapäeval on klikkidele leitud rakendust mitmes erinevas valdkonnas. Bioinformaatikas on kasutatud klikke, et leida sellised geenide grupid, mis avaldavad sarnast pärilikku materjali [5]. Sarnaselt geenide rühmitamisele on klikke rakendatud keemiavaldkonnas selleks, et otsida andmebaasist üksteisele lähedase struktuuriga kemikaale [14]. Lisaks on graafi klikkide leidmise ülesande eeskujul loodud kombinatoorne algoritm, et leida mõne molekulipaari vahel sellised punktid, kus saab toimuda nende molekulide vaheline ühinemine [15].

5.2 Probleemi kirjeldus

Järgmine ülesanne pärineb Stamm-Wilbrandti artiklist [7]. Eesmärgiks on kontrollida, mitmel viisil saab etteantud graafi jaotada K alamgraafiks nii, et iga osa on täisgraaf, kusjuures täisgraafiks loetakse ka tühja ehk tippude ja servadeta graafi. Ehk kui meil on antud

1. graaf $G = (V, E)$, kus V ja E on vastavalt tippude ja servade hulk,
2. positiivne täisarv $K \leq |V|$,

siis kas kehtib järgmine lause: $\exists V = V_1 \cup ... \cup V_k, k \leq K, \forall 1 \leq i \leq k : V_i$ on täisgraaf?

Samas artiklis on välja toodud ülesandele lausearvutusvalem 8, mille kehtestavate väärtustuste kombinatsioonid kirjeldavad neid lausearvutusvalemi muutujaid, kus ülesande on tingimused täidetud. Selle ülesande raames mõistetakse lausearvutusvalemite muutujatena paare $[u, i]$, kus u tähistab tipunumbrit ning i selle graafi järjekorranumbrit, kuhu tipp u kuulub.

Valemi esimene osa ($exactly\_one$ konjunktsioon) jaotab tippude hulga V K erinevaks alamhulgaks ning valemi teine pool (at_most_one konjunktsioon) kontrollib iga alamhulga klikiomadust. Valemi teises pooles genereeritud lausearvutusvalemi muutujate jada annab seega ülevaate ühest võimalikust alamgraafide komplektist. Kliki kontrolli jaoks kasutatakse graafist G täiendi võtmist, mida translaator ei olnud võimeline töötlema enne $notin$ võimaluse lisamist.

$$
\begin{aligned}
X &= \{[v, i] | v \in V, 1 \leq i \leq K\} \\
F &= \bigwedge_{v \in V(G)} (exactly\_one\{[v, i] : 1 \leq i \leq K\}) \\
&\wedge \bigwedge_{1 \leq i \leq K} \bigwedge_{\{u,v\} \notin E; u \neq v} at\_most\_one\{[u, i], [v, i]\}
\end{aligned}
\tag{8}
$$

Enne, kui valemi saab anda translaatorile töötlemiseks ette, tuleb enne teha veel üks teisendus. Nimelt, translaator on üles ehitatud sedasi, et operaator at_most_one

võtab argumendiks ainult ühe tsüklimuutuja, mida võrreldakse tsüklimuutuja kõigi teiste väärtustega. Kuna valem ise aga kasutab sama operaatori juures kahte muutujat korraga, $[u, i]$ ja $[v, i]$, siis tuleb esitada see osa translaatorile vastuvõetaval kujul. Kui meil on kaks tõeväärtust A ja B, millest ülimalt üks võib tõene olla, siis seda võib väljendada kui $\neg(A \wedge B)$, mis on väär vaid siis, kui nii A kui ka B on tõesed.

$$\begin{aligned} f_1 &= \bigwedge_{v \in V(G)} (exactly\_one(x_{v,i} : 1 \leq i \leq K)) \\ f_2 &= \bigwedge_{1 \leq i \leq K} \bigwedge_{\{u,v\} \notin E(G); u \neq v} \neg(x_{u,i} \wedge x_{v,i}) \end{aligned} \tag{9}$$

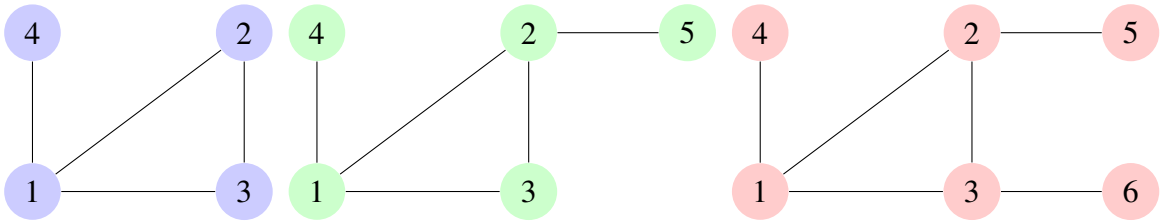
Seega, selle muutuse sisseviimisel hakkab translaator töötlema teisendatud valemit 9, kus paarid $[u, i]$ ja $[v, i]$ on asendatud vastavalt muutujatega $x_{u,i}$ ja $x_{v,i}$ ning at_most_one asemel on nende kahe muutuja konjunktsiooni eitus.

5.3 Probleemi kirjelduse kontrollimine

Kirjelduse kontrollimiseks antakse translaatorile argumendiks valemit 9 sisaldav fail $cliques.tex$ ning graafiargumendiks joonisel 11 kujutatud kolm graafi, mis on lisatud ka repositooriumisse. Translaator käivitatakse iga graafi korral eraldi koos kõikide antud graafi sobilike K argumentidega, kus K võib ülimalt olla graafi tippude arv. Translaatori töö tulemusena genereeritud parameetriteta loogikavalemite kehtestavate väärtustuste arvud on kirjeldatud tabelis 2.

Kehtestavate väärtustuste loendaja eristab õigeid vastuseid ainult lausearvutusmuutuja tasemeni. Seetõttu näiteks kehtestavate väärtuste paari $\{x_1, x_2\}$ loetakse erinevaks paarist $\{x_2, x_1\}$ ehk tagastusväärtuseks on sobivate kehtestavate väärtuste kõikvõimalikud permutatsioonid. Kui aga vaadeldakse sellist alamgraafide komplekti, kus leidub üks või rohkem tühja alamgraafi, siis kehtestaja ei kajasta seda lausearvutusmuutujate jadas, kuna puudub selline paar $[u, i]$, mille jaoks oleks tipp u defineeritud. Sellest tulenevalt võib esineda kehtestavate väärtuste arvu leidmisel tegelikkusest mõningaid erinevusi.

Järgnevalt uuritakse, kas kehtestavate väärtustuste arvud on korrektsed graafi (a) näitel.



(a) Graaf 1 (b) Graaf 2 (c) Graaf 3

Joonis 11. Graafid klikiülesande kirjelduse kontrollimiseks.

Tabel 2. Kehtestaja poolt leitud võimalike klikikombinatsioonide arv.

Graaf \ **K**	1	2	3	4	5	6
(a)	0	4	36	144	-	-
(b)	0	0	24	216	960	-
(c)	0	0	6	192	1620	7680

Juhul $K = 1$ kontrollitakse, mitmel eri moel on võimalik graafi (a) jaotada üheks alamgraafiks nii, et see alamgraaf oleks täisgraaf. Seda graafi on võimalik jaotada üheks alamgraafiks ainult ühel viisil, kus alamgraafiks ongi see graaf ise. Kuna sellise alamgraafi puhul pole tegemist täisgraafiga, siis on võimalike väärtuste arvuks 0.

Juhul $K = 2$ on graafi võimalik jaotada kaheks täisgraafiks nii, kui eemaldada serv $(4, 1)$ või kui eemaldada servad $(1, 2)$ ja $(1, 3)$. Sellisel juhul saadakse kaks alamgraafide komplekti $(\{1, 2, 3\}, \{4\}$ ning $(\{1, 4\}, \{2, 3\})$. Kehtestavate väärtustuste arv 4 tuleneb mõlema jada erinevatest permutatsioonidest. Kaheelemendilisel jadal on kaks erinevat permutatsiooni, mistõttu nende kahe alamhulga komplekti korral ongi võimaluste arv $2 + 2 = 4$.

Juhul $K = 3$ saab jaotada alamgraafide komplektid kaheks selle järgi, kas tipp 4 on samas komponendis tipuga 1 või tipp 4 on üksinda komponent. Esimesel juhul, kus tipp 4 on samas komponendis kui tipp 1, on kaks võimalust luua kolm alamgraafi: $(\{1,4\},\{2\},\{3\})$ ja $(\{1,4\},\{2,3\},\emptyset)$, kus $\emptyset$ on tühi graaf. Teisel juhul, kus tipp 4 on üksinda, saab vaadelda $C(3,2) = 3$ erinevat võimalust, kuidas graafis (a) on jaotatud tipukolmik $\{1,2,3\}$ kaheks alamgraafiks: $(\{1,2\},\{3\},\{4\})$, $(\{1,3\},\{2\},\{4\})$, $(\{1\},\{2,3\},\{4\})$. Viimane võimalik jaotus on $(\{1,2,3\},\{4\},\emptyset)$. Seega kokku on kuus erinevat võimalust, kuidas jaotada joonisel 11 graafi (a) kolmeks osaks, mis annab, kõigi alamgraafide komplektide järjestuste võimalusi arvestades, $6*3! = 6*6 = 36$ kehtestavat väärtust, mis ühtib ka kehtestaja tulemusega tabelis 2.

Juhul $K = 4$ on sisuliselt võimalik antud graafi jaotada, ilma tühjade alamgraafideta, neljaks alamgraafiks nii, kus iga tipp on omaette alamgraafis. Siit osutub, et sellise alamgraafide komplekti erinevate permutatsioonide arv on $4! = 24$. Igale $K = 3$ arutelust saadud kolmeosaliste graafide komplektidele saab juurde lisada veel ühe tühja graafi, et oleks kaetud $K = 4$ tingimused. Lisades nende kuue graafi kombinatsioonid veel juurde, osutub, et võimalikke kombinatsioone on $24 + (6 * 4!) = 24 + 6 * 24 = 168$.

Põhjus, miks tabelis 2 on võimalike klikikombinatsioonide arvuks märgitud 144, on sellepärast, et kehtestaja ei erista permutatsioonide leidmisel ühes alamgraafide jadas tühjasid graafe omavahel. Juhu $K = 4$ korral leidub kaks alamgraafide komplekti $(\{1,4\},\{2,3\},\emptyset,\emptyset)$ ja $(\{1,2,3\}),\{4\},\emptyset,\emptyset)$, mille erinevate permutatsioonide korral esineb sama järjestusega jadasid korduva tühja hulga tõttu. Seepärast on nende kahe alamgraafi komplekti korral permutatsioone poole vähem, 24 asemel 12. Lahutades need kehtestaja jaoks korduvad võimalused, saadakse sama arv kehtestavaid väärtustusi kui kehtestaja, $168 - 12 - 12 = 144$.

Translaatori abil sai joonise 11 graafi (a) korral veendutud, et valem 8 kirjeldab püstitatud ülesannet tõenäoliselt korrektselt. Graafi (a) juhule sarnaselt saab leida väärtused graafidele (b) ja (c), et leida kinnitust valemi korrektsuse kohta ka nende graafide korral.

6 Kokkuvõte

Mitmeid kombinatoorikaprobleeme saab kirjeldada lausearvutusvalemite abil. Antud loogikavalemi kehtestavate väärtustuste leidmisel on võimalik teada saada püstitatud ülesande võimalikud lahendid, milleks ongi valemi kehtestavad väärtustused.

Ülesannete sellise lähenemisviisiga lahendamiseks on loodud lausearvutusvalemite translaator, mis teisendab LaTeXformaadis parametriseeritud sisendvalemi kindla formaadiga parameetriteta lausearvutusvalemiks. Teisendatud valemikuju saab seejärel kasutada mõne kehtestavate väärtuste loendajaga, et leida valemi kehtestavate väärtustuste arv. Antud töö eesmärgiks oli anda ülevaade, kuidas teha läbi ühe programmeerimiskeele täiendamine eelmainitud lausearvutusvalemite translaatori näitel.

Töös kirjeldati, kuidas keele täiendamise raames lisati translaatori eelprotsessorisse täiendav sisendteksti teisendusreegel, mis lihtsustab valemite kirjeldamist ning suurendab kasutajamugavust. Lisaks täiendati translaatori jaoks loodud keele grammatikat, et oleks võimalik anda sisendiks selliseid loogikavalemeid, mis kasutavad mõne etteantud graafi täiendit.

Töö tulemusena täienes lausearvutusvalemite translaator selliselt, et translaator on nüüd võimeline töötlema keerulisemaid loogikavalemeid lihtsustatumal kujul. Selle näitamiseks analüüsiti läbi üks Stamm-Wilbrandti poolt kirjeldatud probleem graafi klikkide kontrollimise kohta, mida varem translaator ei olnud võimeline läbi töötlema. Esialgse translaatori ja tehtud täienduse lähtekood on kättesaadaval lisas 1 täpsustatud repositooriumi kaudu.

Antud töö on edaspidi abiks lugejale, kes soovib saada ettekujutust, kuidas läheneda ühe olemasoleva programmeerimiskeele täiendamisele translaatori ja selle eelprotsessori tasandil. Täiendatud translaatorprogramm aitab viia läbi selliseid teaduslikke eksperimente, kus on teada mõne probleemi rahuldavate seisundite arv, aga puudub nende seisundite täielik kirjeldus.

Viidatud kirjandus

[1] Davis M. Influences of Mathematical Logic on Computer Science. *The Universal Turing Machine: A Half-Century Survey.* Berlin: Kammemer & Unverzagt, 1988, p. 315-327.
https://bit.ly/2VWOlaf (09.05.2019)

[2] Peder, A. Superpositional graphs and finding the description of structure by counting method. TÜ arvutiteaduse instituudi doktoritöö. 2010.
https://dspace.ut.ee/handle/10062/14853

[3] Peder, A., Isotamm, A., Tombak, M. A Meta-compiler for propositional Formulae. *Proc. of Seventh Symposium on Programming Languages and Software Tools: Seventh Symposium on Programming Languages and Software Tools*, Szeged, Hungary, 2001, p. 250−261. http://kodu.ut.ee/~ahtip/LOG_COMP/articles/metacompcyb.pdf (09.05.2019)

[4] Satisfiability Suggested Format. 1993, 8 p.
http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf (09.05.2019)

[5] Ben-Dor, A., Shamir, R. ja Yakhini, Z. Clustering gene expression patterns. *Journal of computational biology*, 6(3-4), 1999, p. 281-297. https://bit.ly/2V9ARDG (09.05.2019)

[6] Luce, R.D. Perry, A.D. A method of matrix analysis of group structure. *Psychometrika*, 1949, p. 95-116. https://bit.ly/2sTHNHd (09.05.2019)

[7] Stamm-Wilbrandt, H. Programming in propositional logic or reductions: Back to the roots (satisfiability). Sekretariat für Forschungsberichte, Inst. für Informatik III, 1993. https://bit.ly/2vMngYE (09.05.2019)

[8] Infotehnoloogia. Sõnastik. Osa 7: Programmeerimine.
https://www.evs.ee/tooted/evs-iso-iec-2382-7-2002 (09.05.2019)

[9] Isotamm A. Translaatorite tegemise süsteem. Tartu: Tartu Ülikooli Kirjastus. 2012.

[10] Zenger, Matthias, and Martin Odersky. "Implementing extensible compilers."*ECOOP Workshop on Multiparadigm Programming with Object-Oriented Languages*, 2001. https://bit.ly/2E1jJKq (09.05.2019)

[11] A000170 - Number of ways of placing n nonattacking queens on an n X n board.
https://oeis.org/A000170 (09.05.2019)

[12] CLion: A cross-platform IDE for C and C++. https://www.jetbrains.com/clion/ (09.05.2019)

[13] Buldas A., Laud P., Villemson J. Graafid. Tartu: Tartu Ülikooli Kirjastus. 2003 http://kodu.ut.ee/~peeter_l/teaching/graafid03s/graafid.pdf (09.05.2019)

[14] Rhodes, N., Willett, P., Calvet, A., Dunbar, J.B. ja Humblet, C. CLIP: similarity searching of 3D databases using clique detection. *Journal of chemical information and computer sciences*, 43(2), 2003, p. 443-448.
https://pubs.acs.org/doi/full/10.1021/ci025605o (09.05.2019)

[15] Kuhl, F.S., Crippen, G.M. ja Friesen, D.K. A combinatorial algorithm for calculating ligand binding. *Journal of Computational Chemistry*, 5(1), 1984, p. 24-34.
https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.540050105 (09.05.2019)

Lisad

I. Lähtekood

Translaatori lähtekood ning sellega seostuvad abifailid on kättesaadaval Githubi repositooriumis aadressil https://github.com/aloaas/logcomp.
Repositooriumis on failid jaotatud järgmiselt:

1. kaustas $queens$ on lippude leidmise ülesandega seotud failid;
2. kaustas $cliques$ on graafi klikkide kontrollimise ülesandega seotud abifailid;
3. kaustas $graphs$ on klikiülesande kontrollimisel kasutatud graafide failid;
4. kaustas $MinGW$ on programmi kompileerimiseks kasutatud kompilaator;
5. kaustas $program$ on täiendatud programm, kus
   (a) translaatori lähtekood asub failis $log32.c$;
   (b) eelprotsessori lähtekood asub failis $lahti.c$;
   (c) $dsat.exe$ on genereeritud väljundi kehtestamiseks;
   (d) $metacomp.bat$ on translaatori ja eelprotsessori käivitamiseks mõeldud skript, millele tuleks anda ette (samas kaustas) töödeldava faili nimi;
   (e) $transcomp.bat$ on mõeldud ainult translaatori käivitamiseks, ilma eelprotsessorita;
6. kokkupakitud failis $LOG\_COMP\_2014.zip$ on programmi esialgne versioon;
7. kokkupakitud failis $forms.zip$ on täiendatud keele grammatikafailid.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Alo Aasmäe**,

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

   Keele täiendamine lausearvutusvalemite translaatori näitel

   mille juhendaja on Ahti Peder

   1.1 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 rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Alo Aasmäe
Tartus, 10.05.2019