TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Siiri Saar

Teek predikaatarvutuse väljendamisülesannete lahenduste kontrollimiseks

Bakalaureusetöö (9 EAP)

Juhendaja: Reimo Palm

Tartu 2017

Teek predikaatarvutuse väljendamisülesannete lahenduste kontrollimiseks

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärk on implementeerida teek predikaatarvutuse väljendamisülesannete lahenduste kontrollimiseks.

Teeki kuuluvad meetodid, mis võimaldavad kontrollida, kas sisend, s.t aritmeetika signatuuris $\sigma = <0, 1; +, *; =>$ kujutatud predikaatarvutuse valem, on korrektsel kujul ja kas valem väljendab etteantud predikaati. Valemi süntaktilise korrektsuse hindamiseks kontrollitakse muuhulgas järgnevat: valem sisaldab vaid etteantud signatuuri kuuluvaid konstant-, funktsionaal- ja predikaatsümboleid, valemis esinevad sulud on tasakaalus, lõppvalemis kasutatavad abipredikaadid on eelnevalt defineeritud, abipredikaadi defineerimisel ei kasutata predikaadi tähises vabu muutujaid, mida abipredikaadi valemis ei esine ja vastupidi – abipredikaadi valemis ei esine vabu muutujaid, mida tähises ei ole välja toodud.

Selleks, et kontrollida, kas valem väljendab etteantud predikaati, kasutatakse käesolevas töös kahte meetodit. Tõesuspuu meetodiga üritatakse kindlaks teha, kas sisestatud valem ja õige valem on samaväärsed, teisisõnu, kas nende kahe valemi ekvivalents on samaselt tõene. Kui tõesuspuu meetod ei anna määratud aja jooksul kindlat tulemust, võetakse kasutusele meetod, mis kontrollib, kas kahe valemi tõeväärtused langevad kokku suurel lõplikul hulgal.

Kuigi valemite samaväärsuse kontroll on algoritmiliselt mittelahenduv, võimaldavad arendatud teegi meetodid anda kindla vastuse laia alamhulga ülesannete korral, tõhustades seega praegust predikaatarvutuse väljendamisülesannete lahenduste kontrollimise töökorraldust Tartu Ülikooli matemaatilise loogika kursustel.

Võtmesõnad:

Matemaatiline loogika, predikaatarvutus, tõesuspuu, signatuur, valemite samaväärsus, vabad ja seotud muutujad, ekvivalents, samaselt tõene valem, üldisus- ja olemasolukvantor

CERCS: P110 (Matemaatiline loogika, hulgateooria, kombinatoorika), P175 (Informaatika, süsteemiteooria)

Library for verifying the solutions of predicate calculus

Abstract:

The goal of this bachelor thesis is to implement a library for verifying the solutions to the provided sentences of the predicate calculus. The library is intended to be used in automatizing the examination of students' answers in the predicate calculus courses in Tartu Ülikool. At present, the lab instructors check the answers manually.

The implemented library contains methods that verify the user input from two aspects – the input is syntactically correct and it expresses the given sentence. Input is deemed syntactically correct if it contains the constant, functional and predicate symbols that are permitted with the arithmetic signature of $\sigma = <0, 1; +, *; =>$, a balanced amount of brackets and the helper predicates defined beforehand. For helper predicates, the program also checks the correct usage of free variables, defined as follows – all free variables used in the predicate are provided as arguments to the predicate and all the predicate's arguments are free variables in the predicate.

To check if the user provided formula is equivalent to a correct formula, the program deploys two methods. First, it applies a method called the truth tree, which consists in analysing the logical structure of the equivalence formula, made up of the user input and the correct answer. The second method consists of evaluating the two formulas with a limited set of natural numbers and checking if the formulas' truth values coincide.

Verifying the equivalence of two formulas is a problem that is known to be algorithmically unsolvable. The two selected methods complement each other – the truth tree method allows determining the equivalence of two formulas, while the brute-force evaluation allows determining their non-equivalence.

Keywords:

Mathematical logic, predicate calculus, truth tree, signature, equivalence, free and bound variables, valid formula, universal quantification, existential quantification.

CERCS: P110, P175

Sisukord

1 Sissejuhatus

Predikaatarvutust õpetatakse Tartu Ülikooli informaatika õppekava tudengitele ainetes „Graafid ja matemaatiline loogika” (MTAT.05.122) ja „Sissejuhatus matemaatilisse loogikasse” (MTAT.05.111). Aines raames toimuvates praktikumides lahendavad tudengid predikaatloogika ülesandeid, mis seisnevad etteantud väidete väljendamises predikaatarvutuse valemitega. Tudengite kirjalikult esitatud lahendusi kontrollivad praktikumijuhendajad käsitsi.

Ülesannete lahenduste kontrollimisel käsitsi on mitmeid puudusi. Selline töökorraldus on aeganõudev ja veaohtlik. Tudengitele antav tagasiside hilineb ning palju juhendajate ressurssi kulub mitte niivõrd sisuliste kuivõrd süntaktilist liiki vigade parandamisele. Viimased hõlmavad vigu, nagu valemis esinevad üleliigsed või puuduvad sulud ning signatuuri mittekuuluvate sümbolite kasutamine, mille kontrollimist on võimalik automatiseerida.

Praktikumijuhendajate kogemus näitab, et väidete väljendamine valmistab paljudele tudengitele raskusi. Teemast paremini aru saamiseks oleks tudengitel vaja harjutada rohkem, kui praktikumijuhendajad käsitsi kontrollida jõuavad. Seetõttu seatakse käesoleva bakalaureusetöö eesmärgiks luua programm, mis võimaldab tudengitel automaatselt kontrollida oma lahendusi.

Kahe valemi, s.t. tudengi sisestatud valemi ja õige valemi samaväärsuse kontroll on algoritmiliselt mittelahenduv probleem, mistõttu saab selle lahendamiseks kasutatud meetoditest oodata ainult osalisi tulemusi. Loodav programm vastab küsimusele, kas kaks valemit on samaväärsed, ühel järgnevatest viisidest – „jah”, „ei”, „ei tea”. Seega ei asenda programm inimkontrollijat, kuid annab tudengitele võimaluse rohkemate ülesannetega harjutada ja pakub väärtuslikku tagasisidet.

Käesolev kirjalik töö täiendab programmi jaoks loodud koodibaasi, andes teoreetilise ülevaate predikaatarvutuse olulisematest mõistetest, programmi sisendi oodatavast kujust ning sisendi töötlemise etappidest, alustades sisendi süntaktilise kontrolliga ja sisendi viimisega sisuliseks kontrollimiseks vajalikule kujule, lõpetades samaväärsuse hindamiseks rakendatavate meetoditega. Tutvustatakse kahe kasutatava meetodi valiku põhimõtet, implementatsiooni ja puudusi. Eraldi peatükk on pühendatud mõlema meetodi testimisele testandmetega, s.o aines MTAT.05.122 tudengite esitatud kodutööde lahendustega.

2 Töö eesmärk

Predikaatloogikat tutvustatakse Tartu Ülikooli informaatika tudengitele ainetes „Graafid ja matemaatiline loogika” ja „Sissejuhatus matemaatilisse loogikasse”, kus tudengitel tuleb muuhulgas väljendada etteantud väiteid, kasutades predikaatarvutuse süntaksit.

Väidete väljendamine on ülesanne, mis valmistab praktikumijuhendajate kogemuse põhjal tudengitele raskusi ja mille lahendamisel teevad tudengid palju vigu. Teisalt, on töökorraldus, kus praktikumijuhendajad kontrollivad ülesannete lahendusi käsitsi, aeganõudev, veaohtlik ja toob paratamatult kaasa selle, et tagasiside andmine tudengitele hilineb.

Selleks, et osaliselt automatiseerida ja tõhustada lahenduste kontrollimist, seatakse käesoleva bakalaureusetöö eesmärgiks implementeerida teek, mis verifitseerib järgnevat:

- kas tudengi sisestatud valem on korrektsel kujul, s.t. valemis kasutatakse ainult signatuuri kuuluvaid sümboleid, avavate ja sulgevate sulgude arv on tasakaalus, predikaat on nii mitme kohaline, kui ülesande püstitus ette näeb;
- kas tudengi sisestatud valem ja õige valem on samaväärsed. Kuna kehtib, et valemid F ja G on samaväärsed parajasti siis, kui valem $F \leftrightarrow G$ on samaselt tõene [4:64], on vaja näidata, et tudengi sisestatud valemi ja õige valemi ekvivalents on samaselt tõene. Valemite samaväärsuse kontrollimiseks kasutatakse kaht meetodit – *tõesuspuu meetodit* ja *väärtuste väljaarvutamise meetodit,* mida tutvustatakse järgnevates peatükkides.

Lisaks teoreetilisele ülevaatele valmib antud bakalaureusetöö raames:

- Java programmeerimiskeeles implementeeritud teek, mida on võimalik integreerida üldisesse, erinevaid kasutajarolle toetavasse arvutiülesannete lahendamise keskkonda;
- Veebikeskkond, mis kasutab arendatud teeki. Veebikeskkond kuvab väiteid ja lubab kasutajal sisestada ja kontrollida valemeid. Kontrolli tulemusena selgub üks kolmest variandist:

1. sisestatud valem ja õige valem on samaväärsed
2. sisestatud valem ja õige valem ei ole samaväärsed
3. sisestatud valem ja õige valem võivad olla samaväärsed („ei tea”).

Süntaktiliselt mittekorrektse sisendi korral osutatakse esimesele mittesobivale sümbolile. Mittesamaväärse valemi korral esitatakse kontramudel – vabade muutujate väärtustus, mil valem on väär.

2.1 Lahendatavad väljendamisülesanded

Programm aktsepteerib valemeid, mis on väljendatud aritmeetika signatuuris σ = <0, 1; +, * ; =>. Valemi tõeväärtust uuritakse interpretatsioonil α, kus $M_\alpha = N$ ja kõiki signatuuri sümboleid interpreteeritakse standardselt.

Kasutajal on võimalik sümboliga ':=' defineerida abipredikaate ning neid lausearvutuse tehetega lõppvastusesse kombineerida. Abipredikaadi täpse lubatud kuju võib leida töö lõpus esitatud ANTLR4 grammatika failist.

Arendatud veebilehel on kasutajal võimalik lisada oma vastusesse ka kommentaare – programm eemaldab sümbolite /* */ vahele jääva teksti enne vastuse edasiandmist teegi meetoditele.

Näiteks väite „Arv x on arvu y suurim algtegur” väljendamisel aktsepteerib programm järgnevaid abipredikaate, ning viimasel real kujutatud lõppvastust:

```
T(x,y):=∃a(x*a=y)
A(x):=∀m∀n(x=m*n->m=1∨n=1)&¬(x=1)   /*   x   on   algarv   */
S(x,y):=∃k(x = y + k) /* x >= y */

T(x,y) & A(x) & ∀u(T(u,y) & A(u) -> S(x,u))
```

3 Predikaatarvutuse mõisted

Käesolevas peatükis antakse ülevaade olulisematest predikaatarvutuse mõistetest.

Signatuur – kolmik $S = <C; F; P>$, kus C on konstantsümbolite, F funktsionaalsümbolite ja P predikaatsümbolite hulk, millesse kuuluvaid elemente on lubatud predikaatarvutuse valemites kasutada. Näiteks predikaatarvutuse valem, mis on väljendatud signatuuris $S = <0, 1; +, *; =>$, võib sisaldada ainult konstantsümboleid 0 ja 1, funktsionaalsümboleid + ja * ning predikaatsümbolit =. [4:52]

Term signatuuris S – term on avaldis, mille väärtuseks on indiviidide piirkonna element. Termid defineeritakse signatuuri S jaoks induktsiooniga järgnevate reeglite abil:

1. Iga indiviidmuutuja on signatuuri S term

2. Signatuuri S iga konstantsümbol on S term

3. Kui f on signatuuri S funktsionaalsümbol ja $t_1, t_2, \ldots, t_n$ on signatuuri S termid, siis $f(t_1, t_2, \ldots, t_n)$ on S term [6]

Olgu meil signatuur S, kuhu kuuluvad konstantsümbol c, ühekohaline funktsionaalsümbol f, kahekohaline funktsionaalsümbol g ja indiviidmuutuja x. Siis on näiteks avaldis $g(f(x), f(c))$ term [4:49].

Predikaatarvutuse valem signatuuris S – predikaatarvutuse valem defineeritakse signatuuri S jaoks induktsiooniga järgnevate reeglite abil:

1. Kui P on signatuuri S n-kohaline predikaatsümbol ja $t_1, t_2, \ldots, t_n$ on termid signatuuris S, siis $P(t_1, t_2, \ldots, t_n)$ on signatuuri S predikaatarvutuse valem.

2) Kui F on signatuuri S predikaatarvutuse valem, siis $\neg F$ on signatuuri S predikaatarvutuse valem.

3) Kui F ja G on signatuuri S predikaatarvutuse valemid, siis $(F \,\&\, G)$, $(F \vee G)$, $(F \rightarrow G)$ ja $(F \leftrightarrow G)$ on signatuuri S predikaatarvutuse valemid.

4) Kui x on indiviidmuutuja ja F on signatuuri S predikaatarvutuse valem, siis $\forall xF$ ja $\exists xF$ on signatuuri S predikaatarvutuse valemid. [4:50]

Elementaarvalem ehk atomaarne valem – predikaatarvutuse valem, mis on koostatud predikaatarvutuse valemi definitsiooni punkt 1 järgi. Kõik ülejäänud predikaatarvutuse valemid koosnevad elementaarvalemitest. [4:50]

Seotud muutuja – muutuja x on valemis A seotud, kui x asub olemasolu- või üldisuskvantori mõjupiirkonnas, s.t. valemit $\forall xA$ või $\exists xA$ moodustavas valemis A. [4:50]

Vaba muutuja – muutuja x on valemis A vaba, kui x ei asu ühegi x-le rakendatud olemasolu- või üldisuskvantori mõjupiirkonnas. Näiteks valemis $\forall x(\exists yA(x,y)\&B(x,y))$ esineb muutuja x valemites A ja B seotuna ning muutuja y valemis A seotuna ja valemis B vabana. [6]

Interpretatsioon – predikaatarvutuse valemit saab interpreteerida erinevatel viisidel. Interpretatsioon on see, millega määratakse indiviidide muutumispiirkond ja antakse signatuuri sümbolitele tähendus [6]. Matemaatiliselt, interpretatsioon on paar $\alpha = (M_\alpha, I_\alpha)$, kus M_α on mittetühi hulk, mida nimetatakse põhihulgaks, ja I_α on interpreteeriv kujutus, mis teisendab

1) iga konstantsümboli hulga M_α mingiks elemendiks;

2) iga n-kohalise funktsionaalsümboli mingiks kõikjal määratud n-kohaliseks funktsiooniks hulgal M_α;

3) iga n-kohalise predikaatsümboli mingiks n-kohaliseks predikaadiks hulgal M_α. [4:53]

Samaselt tõene valem – predikaatarvutuse valem F on samaselt tõene, kui ta on tõene igas interpretatsioonis oma vabade muutujate kõikidel väärtustel. [4:59]

Valemite samaväärsus – valemeid F ja G on samaväärsed, kui nende tõeväärtused on võrdsed igas interpretatsioonis valemite vabade muutujate kõikidel väärtustel. Valemid F ja G on samaväärsed parajasti siis, kui valem $F \leftrightarrow G$ on samaselt tõene. [4:64]

Aritmeetika väljendamise mittelahenduvus – Aritmeetika väljendamine on algoritmiliselt mittelahenduv probleem. See tähendab, et ei leidu algoritmi, mis iga sisendi, s.o. predikaatarvutuse valemi puhul tagastab kindla vastuse küsimusele, kas valem on samaselt tõene või mitte. Aritmeetika väidete mittelahenduvus järeldub peatumise probleemi mittelahenduvusest – kuna väited Turingi masinate kohta on väljendatavad aritmeetika signatuuris ja Turingi masinate peatumise probleem on mittelahenduv [4:125], siis on ka aritmeetika väljendamise probleem mittelahenduv.

4 Sisendi töötlemine

Selleks, et kontrollida, kas sisend on süntaktiliselt korrektsel kujul, ning luua sisendi edasiseks analüüsimiseks vajalik andmestruktuur, kasutatakse programmi nimega *ANTLR v4*.

ANTLR v4 on tööriist, mis loob grammatika faili põhjal parseri ehk süntaksianalüsaatori – programmi, mis tegeleb keele analüüsimisega. Grammatika on kogum reegleid, mis väljendavad sisendi lubatud struktuuri [5:10].

Sisendi töötlemine *ANTLR*'iga koosneb kahest etapist – leksikalisest analüüsist ja parsimisest Leksikalise analüüsi käigus valmistab ANTLRi lekser ette sisendi ANTLRi parserile – grammatikasse salvestatud lekserireeglite alusel töötleb lekser sisendiks saadud sümboleid, viib need märkide kujule (ing. k. *token*) ja jätab välja mittevajalikud sümbolid, nagu tühikud ja reavahetused. Parsimise etapi käigus kontrollib ANTLRi parser, kas sisend sobitub parseri reeglitega ja moodustab märkidest sisendi struktuuri kujutava andmestruktuuri ehk parsepuu (ehk süntaksipuu). Kuna ANTLR lubab ühte grammatikafaili koondada nii lekseri kui ka parseri reegleid, eristatakse neid kirjapildiga – lekseri reeglid kirjutatakse algava või läbiva suurtähega, parseri reeglid aga väiketähtedega. [5:10]

Reeglid kirjeldavad keele süntaksit. Nende abil kontrollitakse, kas sisendlause kuulub grammatikaga määratud keelde või mitte. Sisend kuulub keelde, teisisõnu, sisend on korrektne, kui see sobitub grammatikas kirjeldatud reeglitega.

Järgnevalt kujutatud reeglid on väljavõte loodava programmi tarbeks defineeritud grammatikast. Täieliku grammatika võib leida töö lisast.

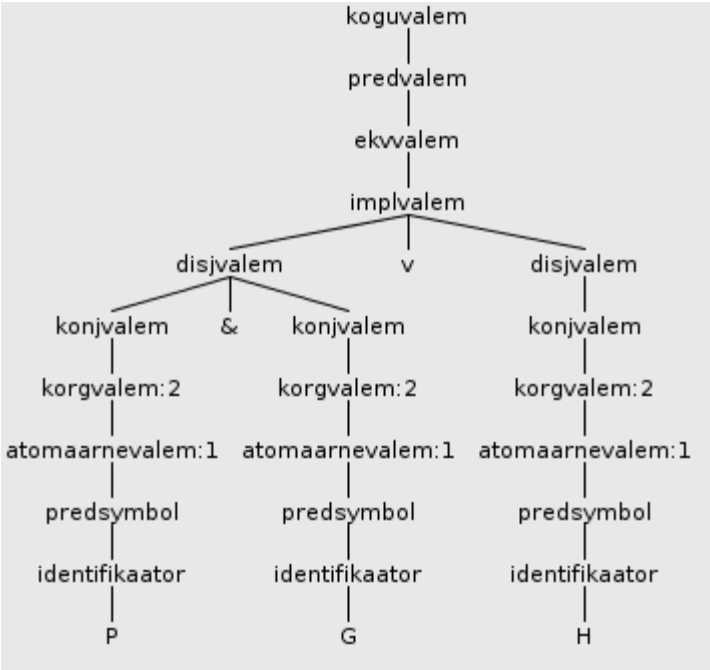
```
grammar Pred;


koguvalem
    : abidef* predvalem;
abidef
    : predtahis ':=' predvalem
    ;
predtahis
    : predsymbol defargumendid*
    ;
```

Parser alustab sisendi sobitamist algreeglist *koguvalem*. Nagu näha, võib koolonist paremale jääva reegli definitsioon koosneda omakorda teistest parseri või lekseri reeglitest või sõneliteraalidest. Koguvalemi reegel defineerib, et sisend võib koosneda nullist kuni mitmest abidefinitsioonist (abipredikaadist) ja predikaatvalemist. Reegel *abidef* täpsustab, et abidefinitsioon peab olema kujul, kus ':=' märgist vasakule jääb predikaatvalemi tähis ja paremale predikaatvalem. Parseri reeglitega luuakse struktuur, mis väljendab valemite vahel loogiliste tehete prioriteeti, viies tugevamini siduvad tehted puus alla poole, nagu näha järgnevalt esitatud parseri reeglite ahelas:

```
predvalem
    : ekvvalem ('~' ekvvalem)*
    ;
ekvvalem
```

```
    : implvalem ('->' implvalem)*
    ;
implvalem
    : disjvalem ('v' disjvalem)*
    ;
disjvalem
    : konjvalem ('&' konjvalem)*
    ;
konjvalem
    : ('¬' | iga | eks)* korgvalem
    ;
```

Joonisel 1 on graafiliselt kujutatud parsepuud, mis väljendab keeles aktsepteeritava näidisvalemi *P&GvH* struktuuri. Parsepuu koosneb reeglite nimedele vastavatest tippudest, mis näitavad, kuidas antud sisendit parsiti. Puu tipus on kõige üldisem reegel, millest analüüsi alustati. Lehtedeks on lekseri reeglitele vastavad sümbolid.



Kuvatõmmis 1: ANTLRi parseri loodud parsepuu

Parsepuu koosneb grammatika reeglite põhjal loodud tippudest (*Context*-tüüpi objektidest), mis on ANTLRisse sisseehitatud klassi *ParseTree* alamklassid.

ANTLRi parsepuu on mahukas ja sisaldab infot sisendi struktuuri ja parsimise kohta. See ei ole valemite edasiseks töötlemiseks mugaval kujul. Seetõttu teisendatakse parsepuu minimalistlikumale, abstraktse süntaksipuu kujule, mis sisaldab sisendi väärtustamiseks ja tõesuspuu koostamiseks olulist infot.

Kasutades ANTLRi parsepuu tipu meetodeid *getChildCount()* ja *getChild(int childIndex)* ning uurides *instanceof* abil parsepuu tippude klasse, läbime parsepuu ja koostame selle põhjal abstraktse süntaksipuu.

4.1 Vigadest teavitamine

Tudengite vastustes esinevad vead võib jagada kahte klassi – sisulised vead ja süntaktilised vead. Sisulised vead hõlmavad mittekorrektsete loogiliste tehete kasutamist, vabade muutujate sidumist kvantoriga ja vastupidi – muutujate, mis peaksid asuma mingi kvantori mõjupiirkonnas, esitamist vabadena. Kõige levinumad süntaktilist laadi vead on aga järgnevad:

1. Kasutatakse sümboleid, mis ei kuulu valemite keelde. Tihti jäetaks tähele panemata signatuuriga kasutada lubatavad konstant-, funktsionaal- ja predikaatsümbolid, ning kujutatakse numbreid, võrratusi, jms. nii nagu need on näha järgnevates testandmete põhjal kogutud valemitest:

*Jagub(x,y) := ∃m(x=m*y) Jagub(x,3)& ¬(Jagub(x,9))* (Näide 1)

*P(x,y) := x > y Q(x,y,z) := z=x*y* (Näide 2)

Sageli esineb tudengite valemites erinevaid keeleväliseid sümboleid, nagu näiteks:

∃y:x=y·(1+1+1) & ¬∃f:x=f·(1+1+1+1+1+1+1+1+1) (Näide 3)

või kasutatakse muid ebastandardseid viise seotud muutujate järjekorra esitamiseks, näiteks:

∃(a, b, c, d): a = y+c & r = y+d & x = a·b &¬(c=0) & ¬(d=0) (Näide 4)

või

∀x,y(∃a,b(P(a,y)&P(b,y) -> Q(a,b,x))) (Näide 5)

2. Sulud ei ole tasakaalus, nagu näiteks järgnevas valemis:

∃y(x=y(1+1+1) & ¬∃z(y=z*(1+1+1))* (Näide 6)

Lahendustes – eriti mitmerealistes ilma abipredikaatideta valemites – üleliigsete või puuduvad sulgude tuvastamine on töömahukas ka kontrollijate jaoks.

Loodava programmi eesmärk on anda tudengile teada sisendis esinevatest süntaktilistest vigadest. Selleks otstarbeks laiendatakse ANTLRi parserisse sisse ehitatud automaatset süntaksivigadest teavitamise süsteemi, väljastades sõnumi ebasobiva sümboli asukoha ja sisu kohta. Asukoha all mõistetakse alates 0st loendatavat positsiooni numbrit sisendtekstis.

Kasutades eeltoodud näiteid, väljastab programm alljärgnevad veasõnumid ja lõpetab töö erindiga:

Positsioonil 32 on lubamatu sümbol '3'. (Näide 1)

Positsioonil 12 on lubamatu sümbol '>'. (Näide 2)

Positsioonil 2 on lubamatu sümbol ':'. (Näide 3)

Positsioonil 13 on lubamatu sümbol ': '. (Näide 4)

Positsioonil 2 on üleliigne ','. (Näide 5)

Positsioonil 33 puudub ')'. (Näide 6)

ANTLR rakendab mitmeid võtteid vigadest taastumiseks, mis võimaldavad parseril pärast süntaksivea leidmist tööd jätkata [5:158]. Kuigi ANTLRi parser võimaldab tänu sellele omadusele väljastada kõiki veateateid korraga, on loodavas programmis valitud pedagoogilistel eesmärkidel teistsugune lähenemine – veakohtadele osutatakse ükshaaval, et tudengitel tekiks sarnaste vigade osas teatav ettenägelikkus ja tähelepanelikkus. Nii

kuvatakse näites 1 kujutatud sisendi kontrollimisel esiteks ainsana veapositsiooni 32, et anda tudengile võimalus iseseisvalt märgata ja parandada ka teist samalaadi viga.

Lisaks süntaktilistele eksimustele valmistab tudengitele raskusi vabade ja seotud muutujate korrektne kasutamine. Väga levinud viga on, et indiviid, mille kohta midagi väidetakse, seotakse olemasolu- või üldisuskvantoriga, nagu näha allolevast valemist väite „x jagub 3ga, aga mitte 9ga" kohta:

$\forall x(Jagub(x,1+1+1) \& \neg(Jagub(x,(1+1+1)*(1+1+1)))$ (Näide 7)

Kontrollides andmebaasi salvestatud õige vastuse vabade muutujate arvu ja tähiseid, väljastab programm ebakõla korral tudengile veateate oodatavate vabalt esinevate muutujate kohta. Näite 7 varal on veateade järgnev: „Esitasid 0-kohalise predikaadi, ootasin aga 1-kohalist predikaati, kus muutuja x esineb vabalt."

Selleks, et aidata tudengitel paremini mõista vabade muutujate tähendust, tehakse abipredikaadi defineerimisele järgmised kitsendused – kõik abipredikaadis esinevad vabad muutujad peavad olema kirjas abipredikaadi tähises, ning kõik tähises nimetatud vabad muutujad peavad vabadena esinema ka abipredikaadis.

Esitades näiteks järgneva abipredikaati sisaldava valemi

$J(x,y,z) := \exists z(x = y * z)\ \ J(x, 1+1+1) \& \neg J(x, 1+1+1+1+1+1+1+1+1)$ (Näide 8),

lõpetab programm töö veateatega „Abipredikaadi definitsiooni kohaselt on muutuja z vaba, kuid abipredikaadis esineb see seotuna või on puudu".

Esitades aga valemi

$J(x,y):=\exists z(x = y * z) \& \neg(k=0)\ J(x, 1+1+1) \& \neg J(x, (1+1+1)*(1+1+1))$ (Näide 9)

kuvatakse tudengile veateadet „Abipredikaadis esineb vabana muutuja k, mida ei ole abipredikaadi tähises kirjas."

Ühtlasi kontrollib programm, kas valemis kasutatav abipredikaat on nimetatud sümboli ja argumentide arvuga eelnevalt defineeritud. Vastasel korral lõpetab programm vastavasisulise veateatega. Sisestades näiteks valemi

$J(x,y):=\exists z(x = y * z)\ P(x, 1+1+1) \& \neg J(x, (1+1+1)*(1+1+1))$ (Näide 10),

väljastab programm teate: „2-kohaline abipredikaat predikaatsümboliga P ei ole defineeritud.".

5 Kasutatavad meetodid

Antud töös implementeeritakse kahe valemi samaväärsuse kontrollimiseks tõesuspuu meetod ja väärtuste väljaarvutamise meetod. Mõlemat meetodit tutvustatakse üksikasjalikult vastavates peatükkides – antud peatükis antakse ülevaade meetodite valiku põhimõttest, ning meetodite rakendamise võimalikest tulemustest.

Implementeeritud meetodid võimaldavad kindlaks teha kahe valemi samaväärsust küll paljudel juhtudel, kuid kuna valemite samaväärsuse kontrollimine on mittelahenduv probleem, mitte alati. Valitud meetodid täiendavad teineteist, kuna tõesuspuu meetod võimaldab kindlalt näidata valemite samaväärsust, väärtuste väljaarvutamine aga valemite mittesamaväärsust.

Esimese võttena rakendatakse tudengi sisestatud valemi ja õige valemi ekvivalentsile tõesuspuu meetodit. Selle meetodiga üritatakse leida väärtustust, mil valem on väär. Kui tehtud eeldus viib analüüsi käigus moodustunud tõesuspuu igas harus vastuoluni, siis oleme tõestanud, et kaks valemit on samaväärsed. Kui puus leidub aga haru, kus vastuolu puudub, ei saa me teha järeldust valemi samaväärsuse ega mittesamaväärsuse kohta. Vastuolu puudumine võib olla põhjustatud sellest, et kaks muidu samaväärset valemit erinevad termide poolest – näiteks on erinev kahe valemi termide osade järjekord. Kuna tõesuspuu meetodit rakendatakse ainult valemi loogilisele osale, uurimata täpsemalt aritmeetikatermide struktuuri, näiteks seda, kas kahe valemi kommutatiivse tehte argumendid on vahetatud, ei tuvasta tõesuspuu meetod kahte samaväärset, kuid erinevate termidega valemit. Samuti võib juhtuda, et tõesuspuu konstrueerimine ei lõppe lõpliku aja jooksul või võtab meetod tõesuspuu tippude eksponentsiaalse plahvatuse tõttu automaatse kontrollimise seisukohast ülemäära kaua aega.

Kui tõesuspuu sisaldab mittevastuolulisi harusid või tõesuspuu meetod ei lõpeta määratud aja jooksul, siis võetakse valemite samaväärsuse hindamiseks kasutusele teine, väärtuste väljaarvutamise meetod. Selle meetodiga kontrollitakse, kas tudengi sisestatud valemi ja õige valemi tõeväärtused langevad kokku suurel lõplikul hulgal. Kui kahest valemist moodustatud ekvivalents ei ole lõplikul kandjal samaselt tõene, siis oleme näidanud, et tudengi sisestatud valem ja õige valem ei ole samaväärsed. Kui ekvivalentsivalem on lõplikul naturaalarvude hulgal väärtustatuna samaselt tõene, siis ei saa me aga teha lõplikku järeldust valemite samaväärsuse ega mittesamaväärsuse kohta. Alljärgnev tabel esitab meetodite võimalikke tulemusi skemaatiliselt.

Tingimus	**Tõesuspuu**	**Väärtuste väljaarvutamine**
Iga puu haru on vastuoluline	Valemid on samaväärsed	-
Leidub avatud haru või puu ei saa valmis määratud ajaga	Ei tea	-
Ekvivalentsivalem ei ole lõplikul kandjal samaselt tõene	-	Valemid ei ole samaväärsed
Ekvivalentsivalem on lõplikul kandjal samaselt tõene	-	Ei tea

6 Tõesuspuu

Üks viis, kuidas kontrollida, kas kaks valemit on samaväärsed, on moodustada tõesuspuu.

Järgnevates alapeatükkides antakse teoreetiline ülevaade tõesuspuu meetodist, kirjeldatakse tõesuspuu implementatsiooni käesoleva bakalaureusetöö tarbeks ning selle meetodi piiranguid.

6.1 Meetodi kirjeldus

Käesolev peatükk on refereeritud õpikust „Sissejuhatus matemaatilisse loogikasse", lk 15-16 [4].

Tõesuspuu on skeem, mis võimaldab valemi loogilist struktuuri uurides kindlaks teha väärtustused, mil valem on tõene või väär.

Tõesuspuu tipus on analüüsitav valem koos tõeväärtusega, mida soovitakse kontrollida. Tipus olevat valemit hakatakse jagama väiksemateks osadeks, lähtudes *elementaarsammudest*. Elementaarsamm on reegel, mis ütleb, millised peavad olema valemi komponentide tõeväärtused selleks, et valemil oleks antud tõeväärtus.

Igal lausearvutuse tehtel ja valemil, mis algab olemasolu- või üldisuskvantoriga, on oma elementaarsammud. Need on esitatud käesoleva peatüki alajaotuses „Elementaarsammud".

Elementaarsammude abil tehakse kindlaks valemi komponentide tõeväärtused. Valemi komponendid kirjutatakse koos järeldunud tõeväärtusega valemi alla. Need on omakorda valemid, mis vajavad edasist analüüsimist. Sellist valemi tükeldamise protsessi korratakse nii kaua, kuni jõutakse valemi väikseimate osadeni, s.o. lausemuutujateni.

Mõned elementaarsammud jagavad tõesuspuu kaheks haruks. Hargnemine tähistab olukorda, kus valemi alamosadel on mitu sobivat tõeväärtust. Näiteks selleks, et valem $F \leftrightarrow G$ oleks tõene, on kaks võimalust – F on tõene ja G on tõene või F on väär ja G on väär. Tõesuspuu hargneb sealkohal kaheks. Valemite edasine analüüs kandub mõlemasse harusse – seni analüüsimata valemid jagatakse elementaarsammude abil väiksemateks tükkideks ja saadud tulemus lisatakse mõlemasse harusse.

Tõesuspuu haru on *lõpetatud*, kui selles harus on kõik valemid jagatud atomaarseteks valemiteks.

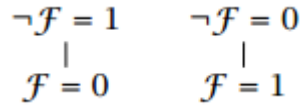
Võib juhtuda, et valemi analüüsi käigus jõutakse mingis harus vastuoluni – olukorrani, kus valem esineb harus nii tõesena kui ka väärana. Sellist haru nimetatakse *vastuoluliseks*.

Tõesuspuu on valmis, kui iga selle haru on kas lõpetatud või vastuoluline.

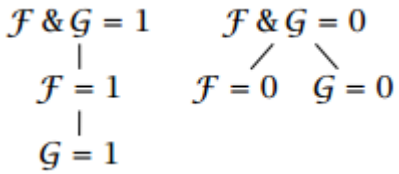
6.1.1 Elementaarsammud

Lausearvutuse tehete ja kvantoriga algavate valemite elementaarsammud on järgnevad [4:16, 61]:

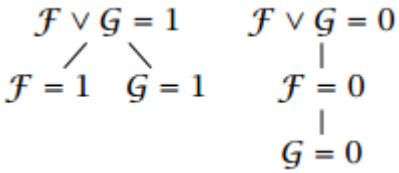
Eitus



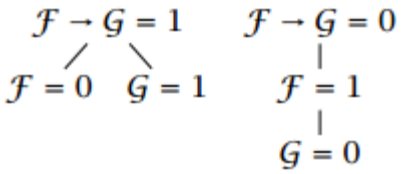
Konjunktsioon



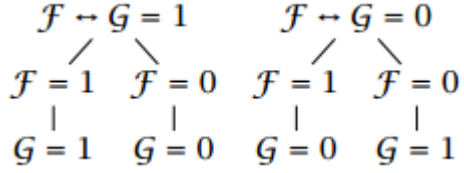
Disjunktsioon



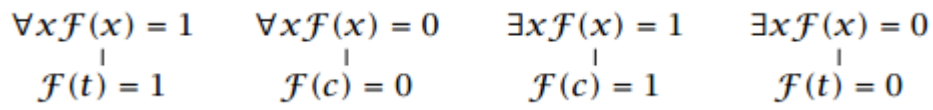
Implikatsioon



Ekvivalents



Kvantoriga valemid



Kvantoreid sisaldavate valemite analüüsimisel tuleb arvestada teatud lisareeglitega. Sõltuvalt tingimusest seisneb kvantoriga valemi analüüsisamm selles, et kasutatakse vaadeldavas puu harus juba olemasolevat termi *t*, või tuuakse sisse uus, vaadeldavas harus seni esinemata sümbol *c*.

Tingimus $\forall xF(x) = 1$ ütleb, et predikaat *F* on tõene iga termi *t* puhul. Seega lisatakse tingimus $F(t) = 1$ tõesuspuusse iga selles harus esineva termi *t* puhul, sealhulgas nende termide puhul, mis tekivad hiljem.

Kui $\exists xF(x) = 0$, siis kehtib väide „On väär, et leidub element, mille korral predikaat *F* on tõene". Teisisõnu, predikaat *F* on väär iga termi *t* puhul. Seega lisatakse tingimus $F(t) = 0$ tõesuspuusse iga selles harus esineva, kaasa arvatud hiljem tekkiva termi *t* jaoks.

Tingimuse $\forall xF(x) = 0$ analüüsimisel tuleb aga tõesuspuusse sisse tuua uus konstantsümbol. Kui $\forall xF(x) = 0$, siis kehtib väide „On väär, et iga elemendi korral on predikaat *F* tõene", ehk teisisõnu, leidub element, mille korral predikaat *F* on väär. Sellisel juhul tuleb kasutusele võtta uus konstantsümbol *c*, kuna puudub alus väita, et element, mille korral *F* on väär, on üks harus eelnevalt esinenud elementidest.

Uus konstantsümbol tuleb sisse tuua ka tingimuse $\exists xF(x) = 1$ analüüsimisel – kui leidub element, mille korral predikaat *F* on tõene, ei anna see meile alust eeldada, et selleks elemendiks on mõni harus varasemalt esinenud term.

Kuna kehtib, et interpretatsiooni põhihulk ei saa olla tühi, siis saab esimese elemendi puusse sisse tuua ka olukorras, kus on vaja analüüsida tingimusi $\forall xF(x) = 1$ või $\exists xF(x) = 0$, kuid vaadeldavas harus ei esine eelnevalt ühtegi termi. Selline olukord ilmneb näiteks valemi $\forall xA(x) \rightarrow \exists xA(x) = 0$ analüüsimisel, kus valemi $\forall xA(x) = 1$ analüüsimisel tuuakse puusse element *a*, nagu näha allolevast tõesuspuust: [4:63]

$$\forall xA(x) \rightarrow \exists xA(x) = 0 \quad (1)$$
$$\forall xA(x) = 1 \quad (2)$$
$$\exists xA(x) = 0 \quad (3)$$
$$A(a) = 1$$
$$A(a) = 0$$

Kui tõesuspuu vaadeldav haru ei sisalda ühtegi termi, kuid võimalik on analüüsida tingimusi $\forall xF(x) = 0$ või $\exists xF(x) = 1$, asutakse mainitud valemeid lahti kirjutama enne tingimusi $\forall xF(x) = 1$ või $\exists xF(x) = 0$.

6.1.2 Näide

Tõesuspuu meetodi selgitamiseks vaatleme järgnevat kvantoreid sisaldavat valemit [4:62]:

$\forall x(A(x) \rightarrow B(x)) \rightarrow (\exists xA(x) \rightarrow \forall xB(x))$

Olgu meie eesmärk näidata, et valem ei ole samaselt tõene.

Tõesuspuu tipus asub uuritav valem väära tõeväärtusega, kuna eeldame, et leidub interpretatsioon, mil valem on väär.

Juurtipus asuva valemi peatehe on implikatsioon. Selleks, et implikatsioon oleks väär, peab implikatsiooni elementaarsammu kohaselt olema tehte vasak pool tõene ja parem pool väär. Lisame need kaks tippu tõesuspuusse.

$$\forall x(A(x) \to B(x)) \to (\exists x A(x) \to \forall x B(x)) = 0 \quad (1)\ \star$$
$$\forall x(A(x) \to B(x)) = 1 \quad ((5), (7))\ \star$$
$$\exists x A(x) \to \forall x B(x) = 0 \quad (2)\ \star$$

Mõlemat valemit on vaja edasi analüüsida, kuna need ei ole elementaarvalemid. Kuna puusse ei ole eelnevalt toodud ühtegi termi, millele valemit $\forall x(A(x) \to B(x)) = 1$ rakendada, ja võttes arvesse, et valemite analüüsimise järjekord ei muuda lõpptulemust, asume analüüsima valemit $\exists x A(x) \to \forall x B(x) = 0$. Analoogselt juurtipus oleva valemiga, on ka siinkohal valemi peatehe väära tõeväärtusega implikatsioon. Seega lisanduvad puuse järgnevad tipud:

$$\exists x A(x) = 1 \quad (3)\ \star$$
$$\forall x B(x) = 0 \quad (4)\ \star$$

Elementaarsammude kohaselt nõuab kummagi lisandunud tingimuse lahtikirjutus uue, puu harus mitte-esinenud konstantsümboli sisse toomist:

$$A(a) = 1$$
$$B(b) = 0$$

Sellega on valem $\exists x A(x) \to \forall x B(x) = 0$ jagatud väikseimateks osadeks – elementaarvalemiteks. Asume analüüsima valemit $\forall x(A(x) \to B(x)) = 1$, kus peatehteks on üldisuskvantori lahti kirjutamine. Kuna antud tingimus ütleb, et iga elemendi x korral kehtib $A(x) \to B(x)$, siis tuleb seda tingimust kasutada puu vaadeldavas harus esinenud iga termi kohta. Kirjutame tingimuse $\forall x(A(x) \to B(x)) = 1$ lahti konstantsümboli a jaoks ja rakendame tõese implikatsioonivalemi elementaarsammu.

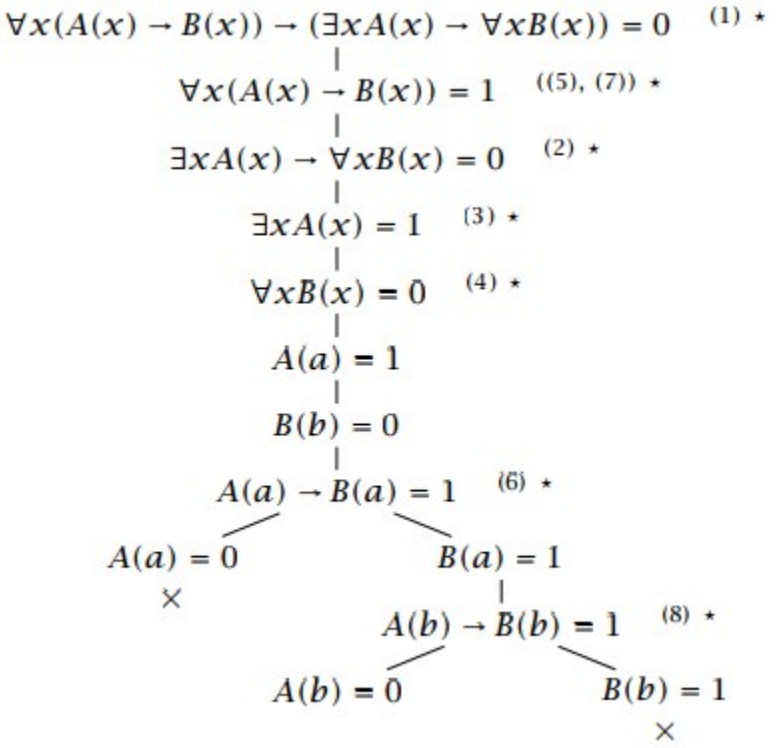
$$A(a) \to B(a) = 1 \quad (6)\ \star$$
$$A(a) = 0 \qquad B(a) = 1$$
$$\times$$

Paneme tähele, et puu vasakus harus olev valem $A(a) = 0$ on vastuolus harus ülal asuva valemiga $A(a) = 1$. Vastuolulises vasakus harus analüüsi ei jätkata.

Järgmiseks väidame tingimust $\forall x(A(x) \to B(x)) = 1$ ka puu harus esinenud termi b kohta, ning kirjutame lahti järeldunud tõese implikatsioonivalemi:

$$A(b) \to B(b) = 1 \quad (8)\ \star$$
$$A(b) = 0 \qquad B(b) = 1$$
$$\times$$

Elementaarvalem $B(b) = 1$ on vastuolus puu harus varasemalt esinenud valemiga $B(b) = 0$. Kuna kõik tõesuspuus esinevad valemid on jagatud elementaarvalemiteks, on analüüs lõppenud ja puu valmis:



Puus esineb kaks vastuolulist haru, mis on märgitud ristiga, ning üks lõpetatud haru, mille lõpus asub leht *A(b) = 0*. Seega leidub interpretatsioon, mil valem on väär. Liigume lõpetatud haru lehest puu juurtipu suunas ja kogume kokku elementaarvalemite tõeväärtused sissetoodud elementidel. Valem on seega väär interpretatsioonis α, kus põhihulgaks on hulk $M_\alpha = \{a, b\}$ ja predikaatsümbolite *A* ja *B* interpretatsiooniks *α* on *A(a) = 1*, *A(b) = 0*, *B(a) = 1*, B*(b) = 0*.

6.2 Implementatsiooni kirjeldus

Tõesuspuu tipp

Nagu eelmises peatükis kirjeldati, moodustub tõesuspuu puu tipus asuva valemi järk-järgulisel jagamisel väiksemateks, samuti tõeväärtusega alamvalemiteks.

Tõesuspuu tippu kujutatakse klassiga *TõesuspuuTipp*. See klass sisaldab vajaliku info hoidmiseks Valem tüüpi valemit ja *boolean*-tüüpi tõeväärtust, kus Valem on ülemklass, millest pärivad kõik loogilised tehted, eitus, olemasolu- ja üldisuskvantor. Igal loogilisel tehtel, s.o. disjunktsioonil, konjunktsioonil, ekvivalentsil ja implikatsioonil on tehte vasaku ja parema poole kujutamiseks kaks Valem tüüpi alluvat, eitusel ja kvantoritel aga üks Valem tüüpi alluv.

Samuti sisaldab iga *TõesuspuuTipu* isend *Optional<Boolean>* tüüpi välja selle kohta, kas tipp on vastuolus mõne oma vanemaga. Kaks tõesuspuu tippu on vastuolulised, kui nende valemid on samad ja tõeväärtused erinevad. Valemi samasuse kontrollimiseks defineeritakse üle Valemi ja Termi *equals* meetodid. *Equals* meetodit rakendatakse valemi komponentidele rekursiivselt kuni jõutakse valemit moodustavate termideni. Kaks valemit on võrdsed, kui nende termid on võrdsed.

Tõesuspuus üles-alla liikumiseks hoitakse igas *TõesuspuuTipu* isendis maksimaalselt kaht sama tüüpi alluvat ja üht vanemat.

Tõesuspuu moodustamine

Tõesuspuu moodustatakse *TõesuspuuTipu* klassi isenditest. Tõesuspuu tipus asub valem, mis on tudengi sisestatud valemi ja õige valemi ekvivalents. Eeldame, et leidub väärtustus, mil ekvivalentsivalem on väär. Tuletame meelde, et meie eesmärk on kindlaks määrata, kas kaks valemit on samaväärsed, ehk teisisõnu, kas analüüsitav ekvivalentsivalem on samaselt tõene. Kui moodustunud tõesuspuu iga haru on vastuoluline, siis on eeldus, et leidub väärtustus, mil valem on väär, ebaõige. Seega on valem igal väärtustusel tõene, ehk teisisõnu, valem on samaselt tõene.

Tippe, mida on vaja elementaarsammudega analüüsida, hoitakse eelistusjärjekorras (*PriorityQueue*). Järjekord kasutab järjestust, mis seab ettepoole tõesuspuu tipud, mis ei tekita hargnemist ja mis ei ole kujul $\forall xF(x) = 1$ ja $\exists xF(x) = 0$. Sellised tipud eemaldatakse järjekorrast varem.

Tõesuspuu tipp võetakse järjekorrast ning asutakse seda analüüsima. Tipus olevale valemile rakendatakse meetodit *reegel()*. See meetod tagastab listi, mis sisaldab peatükis „Elementaarsammud" tutvustatud struktuuri ja tõeväärtusega tõesuspuutippe. Elementaarsammudega tuvastatud tõesuspuutipud lisatakse analüüsimist ootavate tippude järjekorda.

Leitakse analüüsitava tipu lehed – tipud, millel ei ole alamtippe [3:64]. Iga mittevastuolulise lehe alluva(te)ks seatakse elementaarsammudega leitud tõesuspuu tipp/tipud.

Tingimused $\forall xF(x) = 1$ ja $\exists xF(x) = 0$ väidavad vastavalt predikaadi kehtivust ja mittekehtivust iga elemendi korral. Seega tuleb nende kahe tingimuse analüüs sisse tuua iga kord, kui puusse lisatakse uus term. Sel otstarbel implementeeritakse liides *Termikuulaja*, mis tegeleb tingimuste registreerimisega, mida võib tõesuspuus vaja olla mitu korda kasutada. Igal valemil on olemas meetod *getKuulaja*, mis kõikidel juhtudel, v.a. tingimustel $\forall xF(x) = 1$ ja $\exists xF(x) = 0$, tagastab tühja väärtuse (*Optional.empty()*). Üldisus- ja olemasolukvantori valemi klassis kirjutatakse see meetod aga üle – *getKuulaja* loob uue anonüümse klassi, mis implementeerib *Termikuulaja* liidest ja tagastab oodatud tõeväärtusega tõesuspuu tipu, milles kvantoriga seotud muutuja on asendatud täpsustatud konstantsümboliga.

Järjekorrast võetud tõesuspuu tipu analüüsimisel vaadatakse läbi, kas harus ülevalpool asuvate tippude hulgas on valemeid $\forall xF(x) = 1$ ja $\exists xF(x) = 0$. Iga uus term, mis analüüsitava tipuga puusse toodi, kirjutatakse lahti olemasolevate valemite $\forall xF(x) = 1$ ja $\exists xF(x) = 0$ jaoks ning lisatakse tipu iga mittevastuolulise lehe alla ja töötlemist vajavate tippude järjekorda. Järgnevalt kontrollitakse, kas analüüsitav tipp on $\forall xF(x) = 1$ või $\exists xF(x) = 0$. Kui jah, siis lisatakse puusse (ja töötlemist vajavate tippude järjekorda) valemi lahtikirjutus harus varasemalt esinenud termidega. Nende kahe sammu eristamine – harus olemasolevate tingimuste $\forall xF(x) = 1$ ja $\exists xF(x) = 0$ lahti kirjutamine uute termidega ja uue tingimuse $\forall xF(x) = 1$ või $\exists xF(x) = 0$ lahti kirjutamine harus varem esinenud termidega – võimaldab vältida kõikide puus esinevate termide kordamist ja hoida tõesuspuud väiksemana.

Töö lõpetamine

Tõesuspuu kasvatamine katkestatakse, kui tõesuspuus leidub lõpetatud haru. See tähendab, et leidub interpretatsioon, mil valem on väär (või algoritm ei suutnud vastuolu tuvastada aritmeetikatermide erineva struktuuri tõttu). Seega ei saa kontrollitav valem olla samaselt tõene, ja edasise analüüsi võib katkestada. Liikudes lõpetatud haru lehest ülesse kuni esialgse valemini, kogutakse kokku predikaatide tõeväärtused puusse toodud sümbolitel.

Tõesuspuu arendamine lõppeb ka siis, kui kõik moodustunud tõesuspuu harud on vastuolulised. Kuna eeldasime analüüsi alguses, et leidub interpretatsioon, mil valem on väär,

ning jõudsime analüüsi käigus igas harus vastuoluni, siis järelikult ei leidu valemit vääraks muutvat interpretatsiooni. Analüüsitud ekvivalentsivalem on igas interpretatsioonis tõene, ja ekvivalentsivalemit moodustavad alamvalemid seega samaväärsed.

6.3 Tõesuspuu piirangud

Kuna väidete väljendamine on mittelahenduv, ei ole ka valemi analüüsimisel tõesuspuu meetodiga võimalik suvalise sisendi korral lõpliku ajaga kindlaks teha, kas valem on samaselt tõene või mitte. Järgnevad alapeatükid tutvustavad tõesuspuu teoreetilisi piiranguid ja testimisel ilmnenud puudujääke.

Lõpmatult kasvav tõesuspuu

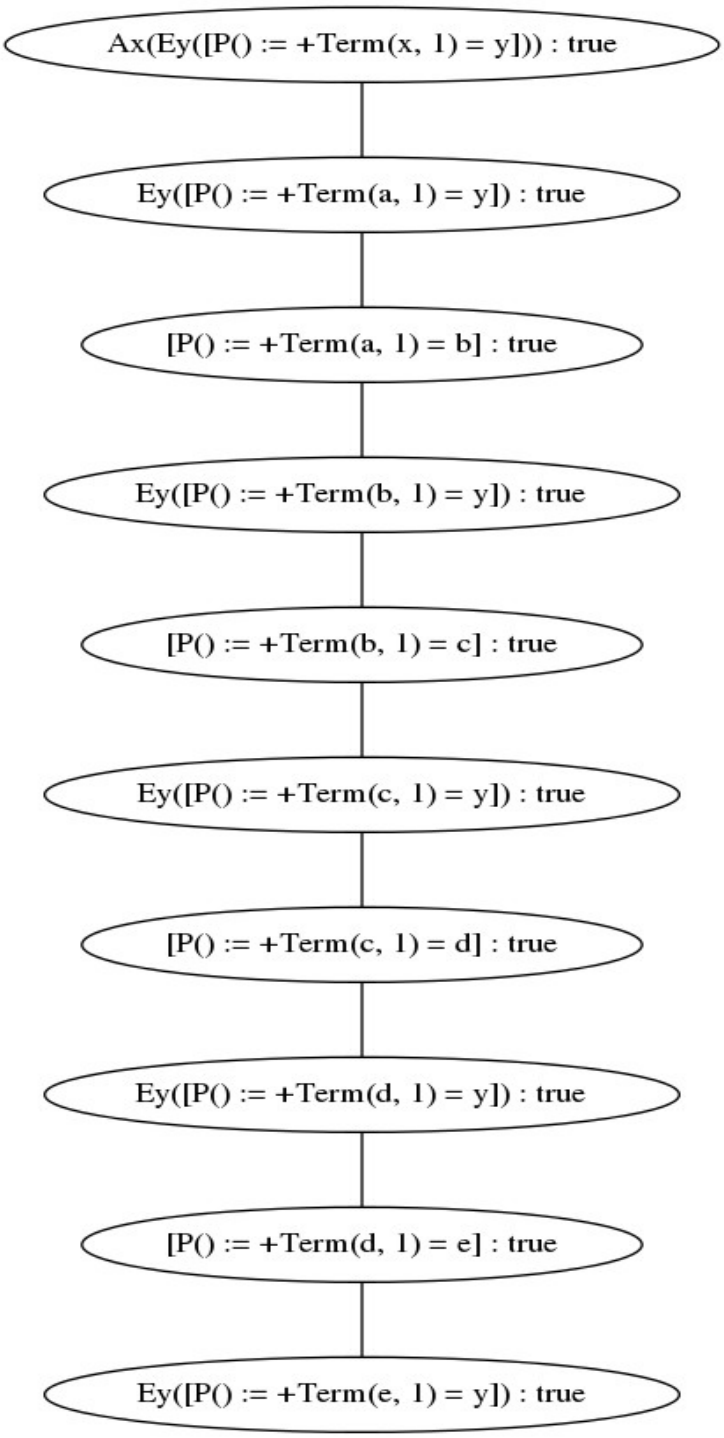
Teatud valemite korral on võimalik, et tõesuspuu moodustamine jääb lõpmatusse tsüklisse.

Olgu meie eesmärk tõesuspuu meetodi abil kontrollida, kas valem $\forall x \exists y P(x, y)$ on kehtestav. Tõesuspuu tipus asub seega valem $\forall x \exists y P(x, y) = 1$. Esimene analüüsisamm nõuab tingimuse $\forall x(...) = 1$ lahti kirjutamist. Peatükis „Elementaarsammud" tutvustatud reegli kohaselt tuleb valem puusse lisada iga puu harus sissetoodud muutuja kohta. Analüüsi alguses ei esine puuharus ühtegi termi. Kuna interpretatsiooni põhihulk ei saa aga olla tühi [4:63], siis võib eeldada, et põhihulk sisaldab mingit elementi. Olgu selleks elemendiks *a*.

Lisame tingimuse $\forall x(...) = 1$ lahtikirjutuse elemendi *a* kohta puusse.

Järgmisena analüüsime tingimust $\exists y P(a, y) = 1$. Elementaarsammu kohaselt kirjutame antud tingimuse lahti uue konstantsümboliga *b*. Kuna predikaat $\exists y P(x,y)$ on tõene iga elemendi *x* korral, siis on see tõene ka elemendi *b* korral. Selle teadmisega lisandub puusse uus tõese tõeväärtusega olemasolukvantorit sisalduv valem, mis jällegi eeldab uue konstantsümboli tutvustamist. Kahe tingimuse lahti kirjutamine jätkub lõpmatult. Tõesuspuu ei saa kunagi valmis ja seega ei saa me teha järeldusi valemi eeldatava tõeväärtuse kohta.

Allolev ekraanipilt näitab lõpmatusse kulgeva valemi $\forall x \exists y P(x, y) = 1$ analüüsi. See on implementeeritud tõesuspuu meetodi väljund.



Tõesuspuu kasvab väga suureks

Isegi kui tõesuspuu moodustamine ei kesta lõpmatu aja, kasvab tõesuspuu teatud kujuga sisendite korral väga suureks ja meetod ei lõpeta töö kontrollimise seisukohast otstarbeka aja jooksul.

Testandmete põhjal ilmneb olukord, kus tõesuspuu järjekorda lisandub väga palju tippe, tudengite vastustes, mis sisaldavad mitmeid kvantoriga ja hargnemisi põhjustavaid valemeid. Olgu tõesuspuu meetodiga kontrollitav valem kujul $\exists x \exists y(\exists w(...) \& \exists v(...)) \sim \exists a \exists b(\exists c(...) \& \exists d(...))$.

Elementaarsammu kohaselt on selline ekvivalentsivalem väär, kui $\exists x \exists y(\exists w(...) \& \exists v(...))=1$ ja $\exists a \exists b(\exists c(...) \& \exists d(...))=0$ või $\exists x \exists y(\exists w(...) \& \exists v(...)) =0$ ja $\exists a \exists b(\exists c(...) \& \exists d(...))=1$.

Kirjutades lahti ekvivalentsi vasakus alluvas valemi $\exists x \exists y(... \& ...)=1$, tuuakse puusse nii mitu muutujat, kui on valemis olemasolukvantoreid. Iga valemis esinevat vaba muutujat ja puu harusse toodud muutujat tuleb väita tingimuse $\exists a \exists b(... \& ...)=0$ lahti kirjutamisel. Kuna see valem sisaldab omakorda hargnemist tekitavat valemit, kandub seni analüüsimata valemite analüüs kõikidesse allpool asuvatesse harudesse. Ekvivalentsivalemi paremas alluvas kulgeb analüüs analoogiliselt – lahti kirjutamist alustatakse valemist $\exists a \exists b(\exists c(...) \& \exists d(...))=1$, mis toob puusse mitmeid uusi muutujaid. Valemi väärust tuleb kõikide sissetoodud muutujate ja vabade muutujate kohta väita valemi $\exists x \exists y(\exists w(...) \& \exists v(...))=0$ lahti kirjutamisel, kusjuures puu hargnemisel kandub seni analüüsimata valemite analüüs igasse alumisse harusse.

Töödeldavate tippude arv järjekorras kasvab eksponentsiaalselt mahuni, mis ei võimalda samaväärsuse kontrolli mõistliku aja jooksul lõpetada.

Esimesel kirjeldatud juhul on töödeldavate tippude järjekorras konstantselt üks tipp, teisel juhul aga väga palju tippe. Selleks, et vältida meetodi väga pikka ja potentsiaalselt lõpmata tööaega, seatakse meetodile ajaline piir – 5 sekundit, mille möödumisel katkestatakse tõesuspuu moodustamine.

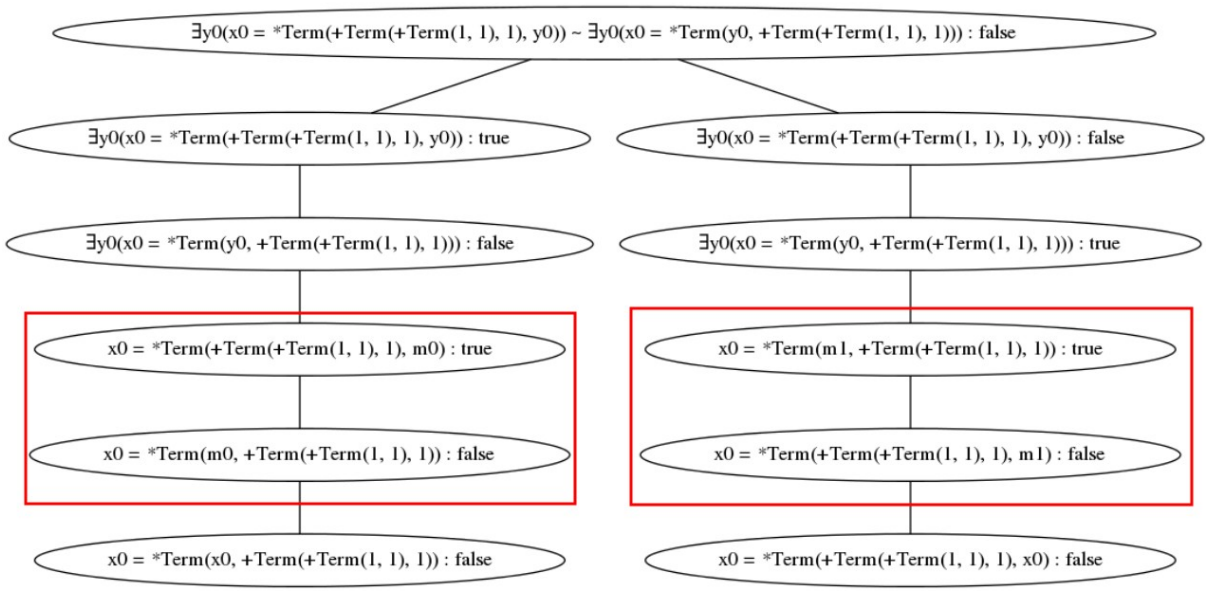
Analüüs piirdub loogilise struktuuriga

Implementeeritud tõesuspuu meetodis on kaks samas tõesuspuu harus asuvat tippu teineteisega vastuolus, kui tipus asuvad valemid on võrdsed ja tõeväärtused erinevad. Kaks valemit on võrdsed, kui valemite alamosad (sh termid) on võrdsed. Taoline süntaktiline võrdlus ei võimalda aga tuvastada samaväärseid valemeid, mis sisaldavad kommutatiivseid või muude binaarsete tehete omadustega tehteid.

Olgu meil nimetatud kitsenduse näitena vaja väljendada väide „x jagub kolmega”. Olgu tõesuspuu tipus valem $Ey(x=(1+1+1)*y) \sim Ey(x=y*(1+1+1)) = 0$.

Kahe valemi samasust kontrollitakse ülekirjutatud *equals* meetodiga, mida kutsutakse rekursiivselt välja valemi alamosade peal. Selleks, et atomaarsed valemid oleksid võrdsed peavad nende vasakud ja paremad termid olema võrdsed. Kui term on binaarne tehe, rakendatakse *equals* meetodit omakorda termi vasakul ja paremal poolel.

Kuvatõmmisel 2 on kujutatud uuritava valemi tõesuspuu. Mõlemas harus on markeeritud kaks tippu, mis on omavahel vastuolus. Vaatleme vasakpoolset haru. Meetod ei tuvasta kahe atomaarse valemi – *x0 = (1+1+1)*m0: true* ja *x0=m0*(1+1+1): false* – vahel vastuolu, kuna atomaarsete valemite paremad pooled koosnevad erinevatest objektidest – ülemises valemis on korrutistermi vasakuks alluvaks liitterm, alumises valemis on korrutistermi vasakuks alluvaks indiviidterm. Programm käsitleb seetõttu neid kahte valemit erinevana. Samal põhjusel ei leita vastuolu parema haru kahe tähistatud tipu vahel.



Kuvatõmmis 2: Näide tõesuspuust, kus vastuolu ei tuvastata

Kirjeldatud kitsenduse tõttu ei tuvasta implementeeritud tõesuspuu meetod samaväärseid valemeid, mis erinevad atomaarse valemi kuju poolest.

6.4 Tõesuspuu meetodi kokkuvõte

Esimese võttena rakendatakse tudengi sisestatud valemi ja õige valemi ekvivalentsile tõesuspuu meetodit. Selle meetodiga üritatakse leida väärtustus, mil valem on väär. Kui tehtud eeldus viib analüüsi käigus moodustunud tõesuspuu igas harus vastuoluni, siis on kaks valemit samaväärsed. Kui puus leidub aga haru, kus vastuolu puudub, ei saa me teha järeldust valemi samaväärsuse ega mittesamaväärsuse kohta.

Testimisel ilmnes kaks tõesuspuuga seotud puudujääki – kuna tõesuspuu meetod piirdub kahe tipu vastuolu tuvastamisel termide süntaktilise kontrollimisega, ei pruugi meetod kindlaks teha tudengite samaväärseid varieeruva kujuga valemeid, nagu $\exists y(x=(1+1+1)*y)$ & $\neg\exists z(x=(1+1+1)*(1+1+1)*z)$ ja $\exists y(x=(1+1+1)*y)$ & $\neg\exists z(x=(1+1+1+1+1+1+1+1+1)*z)$. Samuti näib tõesuspuu plahvatuslikult kasvavat teatud juhtudel, kui analüüsitav valem sisaldab palju kvantoreid ja puu hargnemist põhjustavaid tingimusi. Sellisel juhul ei lõppe tõesuspuu konstrueerimine kontrollimise seisukohast otstarbeka aja jooksul.

7 Väärtuste väljaarvutamise meetod

Tõesuspuud täiendamiseks on antud bakalaureusetöös implementeeritud väärtuste väljaarvutamise meetod.

Kui tõesuspuu sisaldab mittevastuolulisi harusid või tõesuspuu meetod ei lõpeta määratud aja jooksul, siis võetakse valemite samaväärsuse hindamiseks kasutusele teine, väärtuste väljaarvutamise meetod. Selle meetodiga kontrollitakse, kas tudengi sisestatud valemi ja õige valemi tõeväärtused langevad kokku suurel lõplikul väärtuste hulgal. Kui kahest valemist moodustatud ekvivalents ei ole lõplikul kandjal samaselt tõene, siis oleme näidanud, et tudengi sisestatud valem ja õige valem ei ole samaväärsed. Kui ekvivalentsivalem on lõplikul naturaalarvude hulgal väärtustatuna samaselt tõene, ei saa me aga teha lõplikku järeldust valemite samaväärsuse kohta.

7.1 Meetodi kirjeldus

Eelpool tutvustatud definitsiooni järgi on kaks valemit samaväärsed parajasti siis, kui kahe valemi ekvivalents on samaselt tõene. Seetõttu moodustatakse väärtuste välja arvutamiseks tudengi sisestatud valemist ja õigest valemist ekvivalents. Kõik ekvivalentsivalemis esinevad vabad muutujad seotakse üldisuskvantoriga. Seejärel arvutatakse välja ekvivalentsivalemi tõeväärtus lõplikul kandjal, näiteks M = {0, 1, …, 1000}, kusjuures kandja suurus sõltub lõppvalemis esinevate olemasolu- ja üldisuskvantorite maksimaalsest sügavusest.

Kui uuritav valem on kujul $\forall xP(x)$, siis moodustatakse tsükkel üle kõigi M elementide ja kontrollitakse, kas $P(x)$ on igal elemendil tõene. Kui $P(x)$ on mingi kandja elemendi korral väär, väljutakse tsüklist ja tagastatakse *false*.

Kui uuritav valem on kujul $\exists xP(x)$, siis moodustatakse tsükkel üle kõigi M elementide ja kontrollitakse, kas leidub element, mille korral $P(x)$ on tõene. Pärast esimese sellise elemendi leidmist väljutakse tsüklist ja tagastatakse *true*.

Kui uuritav valem koosneb mitmest omavahel loogiliste tehetega ühendatud alamvalemist, nt $P(x) \vee Q(x)$, siis arvutatakse välja $P(x)$ ja $Q(x)$ tõeväärtused väärtusel x, ning rakendatakse tulemustele loogilist tehet.

Näide

Vaatleme näitena, kuidas kontrollitakse kahe valemi samaväärsust, mis väljendavad väidet:

„x jagub 3-ga, aga mitte 9-ga” (Väide 1)

Olgu õige valem kujul

$\exists y(x=(1+1+1)*y) \wedge \neg\exists z(x=(1+1+1)*(1+1+1)*z)$

ning tudengi sisestatud valem kujul

$\exists z(x = (1+1+1)*z) \wedge \forall z\neg(x=(1+1+1+1+1+1+1+1+1)*z)$.

Väärtustatav valem on seega kujul

$\forall x[(\exists y(x=(1+1+1)*y) \wedge \neg\exists z(x=(1+1+1)*(1+1+1)*z)) \sim (\exists z(x = (1+1+1)*z) \wedge \forall z\neg(x=(1+1+1+1+1+1+1+1+1)*z))]$

Kõige välimine tsükkel jookseb üle muutuja x vahemikus 0, 1, .., kandja suurus. Ekvivalentsivalemi tõeväärtuse leidmiseks tuleb kindlaks teha selle vasaku ja parema alluva tõeväärtused. Ekvivalentsivalemi vasak ja parem alluv on mõlemad konjunktsioonid, mille

tõeväärtuste leidmiseks on vaja tuvastada omakorda nende vasak- ja paremalluvate tõeväärtused. Vaatleme näiteks vasakpoolset konjunktsiooni $\exists y(x=(1+1+1)*y)$ & $\neg\exists z(x=(1+1+1)*(1+1+1)*z)$. Selle tõeväärtuse leidmiseks moodustatakse kõigepealt tsükkel 0, 1 ..., kandja suurus üle tema vasaku alluva $\exists y(x=(1+1+1)*y)$. Jõudes rekursiivselt atomaarse valemini $x=(1+1+1)*y$, on x'i ja y'i väärtused välimiste tsüklitega määratud, ning leitakse atomaarse valemi tõeväärtus. Kui konjunktsiooni vasaku alluva tõeväärtus on kindlaks tehtud, korratakse sama protsessi konjunktsiooni parema alluva peal. Kui on selgeks tehtud ekvivalentsi vasaku alluva tõeväärtus, asutakse analoogiliselt väärtustama selle paremat alluvat.

Kandja suurus

Nagu eelpool mainitud, on kandja suurus seatud sõltuvusse valemis esinevate olemasolu- ja üldisuskvantorite maksimaalsest sügavusest, kuna mitmest kvantorist koosnevate valemite puhul on seesmiste tsüklite arv võrdne kvantorite arvuga ja meetodi tööaeg maksimaalselt (kandja suurus)^(kvantorite arv). Vaatleme näiteks samaselt väära valemit $\exists x\exists y\exists z\exists w(x+y+z+w=x+y+z+w+1)$. Selle väärtustamise pseudokood näeks välja järgnev:

```
for(x=0; x < m; x++) {
  for(y=0; y < m, y++){
    for(z=0; z < m; z++){
      for(w=0; w < m; w++){
      }
    }
   }
 }
```

Kuna tegemist on olemasolukvantoriga, üritatakse leida kogu kandja ulatuses väärtustust, mil valem on tõene. Kui kõige seesmine tsükkel muutujaga w jõuab maksimaalse väärtuseni, suurendatakse z'i ning kõige seesmise tsükli väärtustamine hakkab algusest peale. Nii kujuneb väljakutsete arvuks (kandja suurus)^4.

Selleks, et tagada meetodi mõistlik tööaeg (u 5 sekundit), määratakse kandja suurus seega vastavalt maksimaalsele kvantorite sügavusele valemis.

Kontranäide

Kui tudengi sisestatud valem ja õige valem ei ole lõplikul kandjal arvutatuna samaväärsed, tagastatakse kontramudel – vabade muutujate väärtustus, mil kahe valemi tõeväärtused on erinevad. Olgu meil näiteks vaja väljendada eespool tutvustatud väidet 1. Olgu tudengi valem kujul

$\exists m\exists n(((1+1+1)*m=x)\&\neg((1+1+1)*(1+1+1)*n=x))$

ja õige valem kujul

$\exists y(x = y + y + y)$ & $\neg\exists z(x = z + z + z + z + z + z + z + z + z)$

Juba kõige välimise tsükli esimesel iteratsioonisammul (x=0) ilmneb, et kahel valemil on erinevad tõeväärtused. Meetod lõpetab töö ja tagastab tudengile kontranäite kujul „Sinu vastus on väärtustusel {x=0} tõene, aga peaks olema väär”.

7.2 Implementatsiooni kirjeldus

Tõeväärtuse leidmiseks implementeerib iga valem *boolean*-tüüpi tagastusväärtusega meetodi *väärtusta*, mille argumendiks on indiviidmuutujaid ja nende võimalikke väärtusi sisaldav sõnastik (*Map*) ja kandja suurus. See meetod kutsutakse rekursiivselt välja valemi alamosade peal. Selleks, et leida näiteks valemi *Q(x) v R(x)* tõeväärtust, tehakse esmalt kindlaks valemi vasaku ja parema alluva tõeväärtused ja rakendatakse saadud tulemustele disjunktsiooni.

Kui valem on kujul *AxQ(x)* või *ExQ(x)*, siis moodustatakse üle kontrollitavate väärtuste hulga tsükkel, mille igal iteratsiooni sammul väärtustatakse seotud indiviidmuutuja konkreetse väärtusega, mis lisatakse sõnastikku, ja kontrollitakse rekursiivselt, kas valem on sellel väärtustusel tõene. Üldisuskvantorit sisaldava valemi puhul katkestatakse väärtustamine ja tagastatakse *false*, kui valem on kandja mingil väärtustusel väär. Vastasel juhul tagastatakse tsüklist väljudes *true*. Olemasolukvantori sisaldava valemi puhul katkestatakse väärtustamine ja tagastatakse *true* juhul, kui valem on kandja mingil väärtustusel tõene. Vastasel juhul tagastatakse tsüklist väljudes *false*.

Termide väärtuste välja arvutamiseks implementeerib iga term *double*-tüüpi tagastusväärtusega meetodi *väärtusta*, mille argumendiks on valemi edastatud sõnastik. Korrutamist ja liitmist sisaldava avaldise korral kutsutakse *väärtusta* meetodit rekursiivselt välja avaldise paremal ja vasakule poolel, ja rakendatakse tagastatud väärtustele vastavat binaarset tehet.

8 Testandmed ja testimise tulemused

Käesolevas peatükis antakse ülevaade tõesuspuu meetodi ja väärtuste väljaarvutamise meetodi testimise tulemustest.

Meetodite testimiseks kasutati tudengite vastuseid, mida nad esitasid aine MTAT.05.122 väidete väljendamise kodutöö raames.

Õige valemina on välja toodud maksimaalse hinde teeninud suvaliselt valitud tudengi vastus.

Ebaõige sisendi korral on esitatud programmi tööd lõpetava erindi nimi ning veateade.

„-" tähistab olukorda, kus tõesuspuu meetod tuvastas kahe valemi samaväärsuse või programm lõpetas töö erindiga ega jõudnud seega teise meetodi rakendamiseni.

Testandmete osas tasub märkimist, et tegemist on kodutööde väidetega, mis on praktikumis harjutatust keerulisemad ja mahukamad. Seetõttu on ka pakutavad lahendused oma kujult keerukamad, ja toovad kaasa peatükis „Tõesuspuu piirangud" kirjeldatud tõesuspuu tippude eksponentsiaalse plahvatuse. See põhjustab olukorra, et mitmed testjuhtumid annavad samaväärsete valemite korral tõesuspuu meetodiga vastuseks „ei tea". Praktikumides alustatakse lihtsamate valemitega, mille termidel ei ole eriti palju samaväärseid kujusid, ja tõesuspuu meetod annab seega sagedamini tulemuse.

8.1 Väide 1

Väide: „x jagub 3-ga, aga mitte 9-ga"

Õige näidisvalem: ∃y(x=(1+1+1)*y) & ¬∃z(x=(1+1+1)*(1+1+1)*z)

Valem	Tõesuspuu meetod	Väärtuste väljaarvutamine
∃y∀z(x = (1+1+1)*y & ¬(x = (1+1+1)*(1+1+1)*z))	On samaväärne	-
∃a(x = (1+1+1)*a) & ∀b¬(x=(1+1+1)*(1+1+1)*b)	On samaväärne	-
∃z(x=(1+1+1)*z) & ∀z¬(x=(1+1+1+1+1+1+1+1+1)*z)	Ei tea	Ei tea
∀x∃z(x = z * (1+1+1) & ¬(x = z * (1+1+1)*(1+1+1)))	predmoodul.erindid. ErinevIndiviididdeArv Väljastatakse: „Esitasid 0-kohalise predikaadi, ootasin aga 1-kohalist predikaati, kus muutuja x esineb	-

	vabalt.”	
∃z(x=(1+1+1)*z & ¬(x=(1+1+1+1+1+1+1+1+1)*z))	Ei tea	Ei ole samaväärne Väljastatakse: „Sinu vastus on väärtustusel {x=9} tõene, aga peaks olema väär.”
∃y((1 + 1 + 1) * y = x) & ∃z¬((1 + 1 + 1) * (1 + 1 + 1) * z = x)	Ei tea	Ei ole samaväärne Väljastatakse:: „Sinu vastus on väärtustusel {x=0} tõene, aga peaks olema väär.”
Jagub(x,y) := ∃z((x=z*y) & ¬(z=0)) Jagub(x,3) & ¬Jagub(x,9)	predmoodul.erindid.-SyntaksiViga Väljastatakse: „Positsioonil 43 on lubamatu sümbol '3'.”	-

8.2 Väide 2

Väide: „x on kahe y-st suurema arvu korrutis”

Õige näidisvalem: ∃q∃p(∃z(y+z+1=q) & ∃w(y+w+1=p) & (x=q*p))

Valem	Tõesuspuu meetod	Väärtuste väljaarvutamine
Q(x, y) := ∃d(y + d = x & ¬(d = 0)) ∃z∃w(Q(w, y) & Q(z, y) & x = w * z)	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei tea
Suurem(x,y) := ∃m(y+m=x)& ¬(m=0) ∃a∃b(x=a*b & Suurem(a,y) & Suurem(b,y))	predmoodul.erindid.-VaarVabadeMuutujate-Esinemine Väljastatakse: „Abipredikaadis esineb vabana muutuja m, mida ei ole abipredikaadi tähises kirjas.”	-
∃a∃b(∃n(y+n=a) & ∃n(y+n=b) ->	Ei tea	Ei ole samaväärne

x=a*b)	TestTimedOut-Exception: test timed out after 5000 milliseconds	Väljastatakse: „Sinu vastus on väärtustusel {x=0, y=0} tõene, aga peaks olema väär.”
∃z∃w((x = z * w) & ∃a((y + a = z) & ¬(a = 0)) & ∃b((y + b = w) ¬(b = 0)))	predmoodul.erindid.-SyntaksiViga Väljastatakse: „Positsioonil 63 on lubamatu sümbol '¬'. Ootasin üht järgnevast: {'->', '~', '&', ')', '∨'}.”	-
(x = z * w) & ∃k∃l((y + k + 1 = z) & (y + l + 1 = w))	predmoodul.erindid.-ErinevIndiviidideArv: Väljastatakse: „Esitasid 4-kohalise predikaadi, ootasin aga 2-kohalist predikaati, kus muutujad x, y esinevad vabalt.”	-

8.3 Väide 3

Väide: „x ei jagu ühegi y-st väiksema algarvuga”

Õige näidisvalem:

A(x) := ∀y∀z(x = y * z -> y = 1 ∨ z = 1) & ¬(x = 1)

J(x,y) := ∃z(x=y*z) & ¬(y=0)

V(x,y) := ∃z(x + z = y & ¬(z = 0))

∀z(A(z) & V(z,y) -> ¬J(x,z))

Valem	Tõesuspuu meetod	Väärtuste väljaarvutamine
∀w(∀o∀p(w = o * p -> o = 1 ∨ p= 1) & ¬(w = 1) & ∃q(w+q+1=y) -> ¬∃z(x=w*z) & ¬(w=0))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei tea
¬∃z∃w((x = z * w) & ∃k((z + k = y) & ¬(k = 0)) & ∀a∀b(z = a * b -> a = 1 ∨ b = 1) & ¬(z = 1))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei tea

R(x) := ∀y∀z(x = y*z -> y=1 ∨ z=1) & ¬(x=1) V(x,y) := ∃z(x+z=y & ¬(z=0)) ∀r∀z(¬(x=r*z)-> ((R(z)&V(z,y)∨(R(r)&V(r,y)))))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei ole samaväärne Väljastatakse: „Sinu vastus on väärtustusel {x=0, y=0} väär, aga peaks olema tõene.”
¬∃z∃w((x = z * w) & ∃v((z + v = y) & ¬(v = 0)) & ∀a∀b(z = a * b -> a = 1 ∨ b = 1) & ¬(z = 1)	predmoodul.erindid.-SyntaksiViga Väljastatakse: „Positsioonil 92 puudub ')'.”	-
P(x) := ∀y∀z(x = y * z -> y = 1 ∨ z = 1) & ¬(x = 1) ¬∃z(P(z) & z < y & ∃w(x = w * z))	predmoodul.erindid.-SyntaksiViga Väljastatakse: „Positsioonil 65 on lubamatu sümbol '<'.”	-

8.4 Väide 4

Väide: „x < y ning arvude x ja y vahel leidub täpselt üks algarv”

Õige näidisvalem:

A(x) := ∀y∀z(x=y*z -> y=1 ∨ z=1) & ¬(x=1)

V(x,y) := ∃z(x+z+1=y)

∃a(V(x,a)&V(a,y)&A(a)& ∀b(V(x,b) & V(b,y) & A(b) -> b=a))

Valem	Tõesuspuu meetod	Väärtuste väljaarvutamine
V(x,y) := ∃z((x + z = y) & ¬(z = 0)) A(x) := ∀y∀z((x = y * z) -> (y = 1) ∨ (z = 1)) & ¬(x = 1) V(x,y) & ∃s(V(x,s) & V(s,y) & A(s) & ∀t(A(t) & V(x,t) & V(t,y) -> t = s))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei tea
A(x) := ∀y∀z(x = y * z -> y = 1 ∨ z = 1) & ¬(x = 1) V(x,y) := ∃n(x+n+1=y)	predmoodul.erindid.-SyntaksiViga Väljastatakse: „Positsioonil 105 on	-

∃a(V(x,a) & V(a, y) & A(a) & ∀b(x, b) & V(b, y) & A(b) -> b = a))	lubamatu sümbol ','.”	
A(x) := ∀y∀z(x = y * z -> y = 1 ∨ z = 1) & ¬(x = 1) V(x, y) := ∃z(x + z = y & ¬(z = 0)) ∃z(V(x, z) & V(z, y) & A(z)) & ¬∃q(V(x, q) & V(q, y) & A(q) & ¬(q = z))	predmoodul.erindid.-ErinevIndiviidideArv Väljastatakse: „Esitasid 3-kohalise predikaadi, ootasin aga 2-kohalist predikaati, kus muutujad y, x esinevad vabalt.”	-
W(x,y) := ∃z((x + z = y) & ¬(z=0)) P(x) := ∀y∀z(x = y * z -> y = 1 ∨ z = 1) & ¬(x = 1) W(x,y) & ∃a(W(x, a) & W(a, y) & P(a) -> ∀b(W(x, b) & W(b, y) -> ¬P(a)))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei ole samaväärne Väljastatakse: „Sinu vastus on väärtustusel {x=0, y=1} tõene, aga peaks olema väär.”
Algarv(x) := ∀y∀z(x = y*z -> y = 1 ∨ z = 1) & ¬(x = 1) Algarvud(x,y) := ∃m(∃k(m = x + k + 1) & ∃k(y = m+ k + 1) & Algarv(m)) ∃m(y=x+m+1) & ∃z(∃m(z = x+m+1) & ∃m(y=z+m+1) & Algarv(z) & ¬Algarvud(x,z) & ¬Algarvud(z,y))	Ei tea TestTimedOutException: test timed out after 5000 milliseconds	Ei tea

9 Teegi kirjeldus

Java programmeerimiskeeles kirjutatud teek on kättesaadav aadressilt https://github.com/siiri1307/predicate-calculus-program. Projekti haldamiseks ja ehitamiseks kasutatakse Maveni nimelist Java tööriista. Teegi integreerimiseks Maveni projekti lisada `pom.xml` faili, elementi `<dependencies>` järgmine lõik:

```
<dependency>
  <groupId>com.siiri.saar</groupId>
  <artifactId>pred-logic-lib</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

Kahe valemi samaväärsuse hindamiseks luua uus klassi Kontroll isend. Argumentidena anda konstruktorile ette kaks *String*-tüüpi objekti, millest esimene parameeter tähistab kasutaja pakutud vastust ja teine õiget vastust. Kontrolli objekti loomisel käivitatakse meetodid, mis kontrollivad ANTLR 4 abil sisendi süntaktilist korrektsust, viivad sisendid abstraktse süntaksipuu kujule, moodustavad kahest sisendist ekvivalentsivalemi, ning kontrollivad valemite samaväärsust tõesuspuu meetodil ning juhul, kui tõesuspuuga ei tuvastata kahe valemi samaväärsust, ka väärtuste väljaarvutamise meetodil. Alljärgnev rida on näide Kontrolli klassi initsialiseerimisest:

```
Kontroll kontroll = new Kontroll(pakkumineKommentaarideta, oigeVastus);
```

Kontrollimise tulemusena tagastatakse *int*-väärtus, millele on võimalik ligi pääseda Kontrolli objekti meetodiga `getKontrolliTulemus()`. Võttes arvesse töö püstitust leida vastus küsimusele, kas kaks valemit on samaväärsed, on selle tähendus järgnev:

Täisarvuline väärtus	Tähendus

0	Ei, kaks valemit ei ole samaväärsed. Kontramudeli saamiseks kasutada järgneva signatuuriga Kontrolli objekti meetodeid:
	`Map<Character,Integer> tagastaKontramudel();`
	`String kontraNaideStringina();`

1	Jah, kaks valemit on samaväärsed.
3	Ei tea. Tõesuspuu meetodiga ei tuvastatud, et kaks valemit on samaväärsed. Väärtuste väljaarvutamisega ei selgunud, et kaks valemit ei ole samaväärsed, s.t. kahe valemi väärtustamisel täpsustatud lõplikul naturaalarvude hulgal ei leitud väärtustust, mil kahe valemi tõeväärtused on erinevad.

Tabel 1: kontrolli() meetodi võimalikud tagastused

Kontrolli isendi loomisel võib programm visata järgnevaid erindeid:

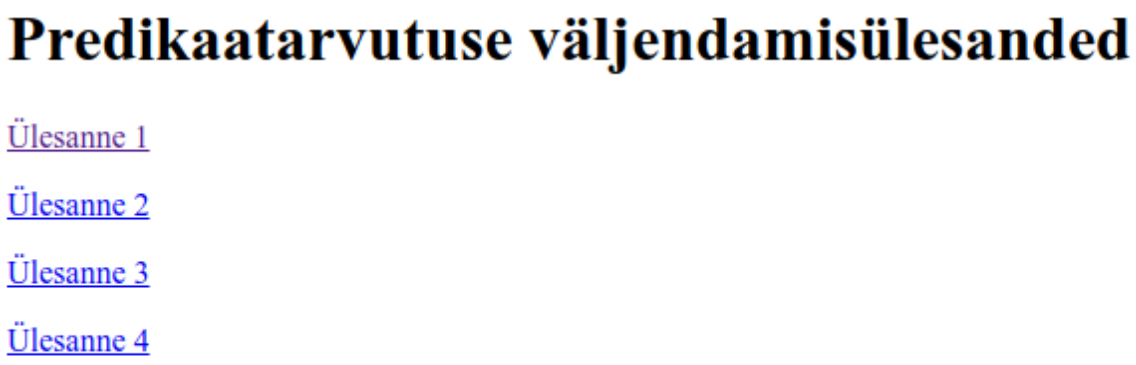
Erind	Tähendus
AbiValemEiOleDefineeritud	Lõppvastuses esitatud abipredikaat ei ole nimetatud tähise või argumentide arvuga eelnevalt defineeritud
ErinevIndiviididdeArv	Kasutaja sisestatud valemi ja õige valemi vabad muutujad on erinevad
IOException	Java I/O erind
SyntaksiViga	Sisendsõne ei sobitu ANTLRi grammatika failis defineeritud reeglitega
VaarVabadeMuutujateEsinemine	Abipredikaadi tähises on välja toodud muutuja, mis abipredikaadis ei esine vabalt või on puudu, või abipredikaadis esineb vabana muutuja, mida ei ole abipredikaadi tähises välja toodud.

Tabel 2: Kontrolli objekti loomisel visatavad erindid

9.1 Näidisveebirakendus

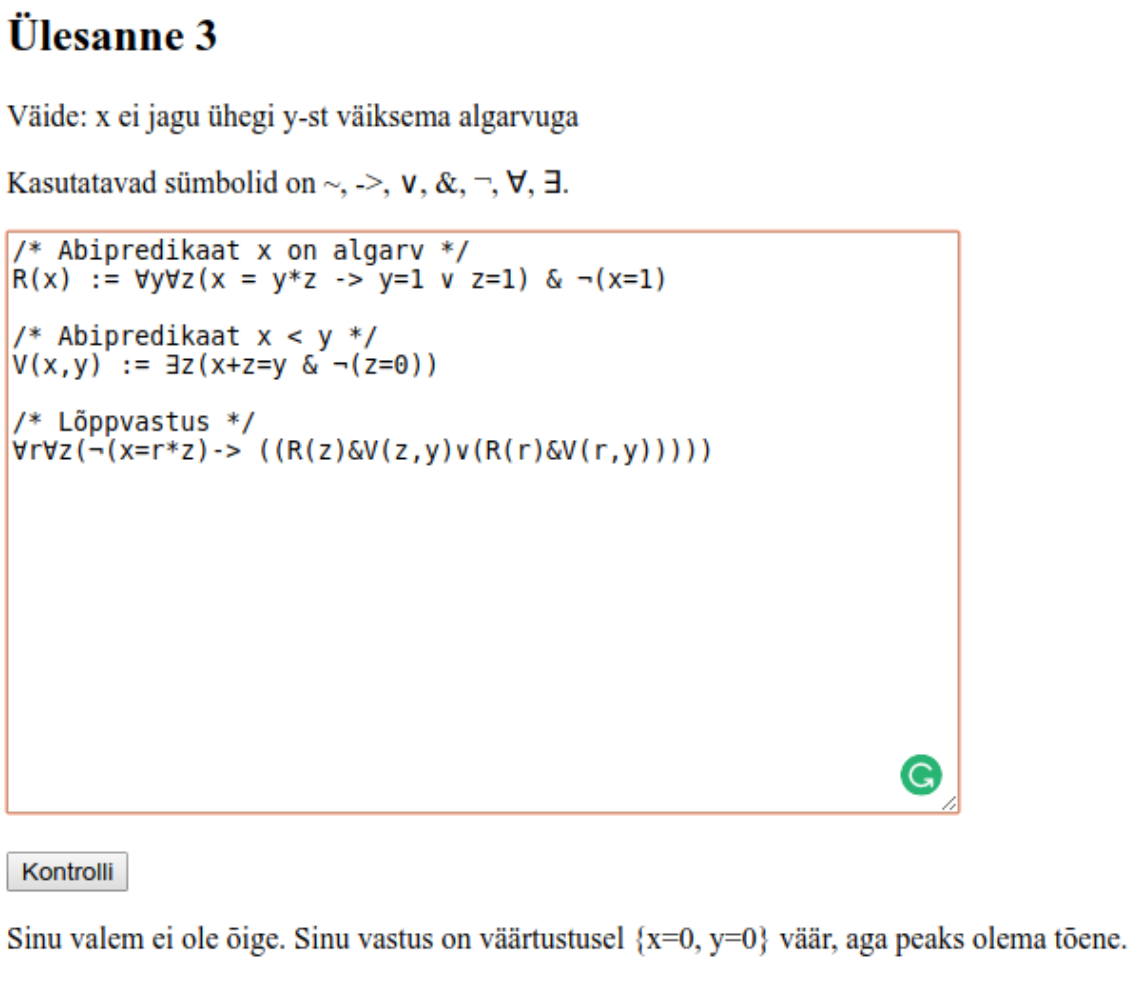
Kuna käesoleva bakalaureusetöö eesmärk on osaliselt automatiseerida predikaatarvutuse väljendamisülesannete lahenduste kontrollimist ja võimaldada tudengitel valemitega rohkem katsetada, kui praegune lahenduste käsitsi kontrollimise korraldus lubab, on teek integreeritud lihtsa kasutajaliidesega veebirakendusse. Rakendus on ajutiselt kättesaadav aadressilt https://pred-valjendamisylesanded.herokuapp.com/.

Rakenduse esilehel kuvatakse ülesannete nimekirja.



Kuvatõmmis 3: Veebirakenduse esileht

Klõpsates ülesandel, avaneb vaade, kus kuvatakse ülesande väidet. Tudengil on võimalik tühja tekstialasse sisestada valemeid, mis on peatükis „Lahendatavad väljendamisülesanded" kirjeldatud kujul. Samal lehel kuvatakse ka kontrolli tulemust.



Kuvatõmmis 4: Veebirakenduse ülesande vaade

10 Ülevaade predikaatarvutuse veebiprogrammidest

Antud bakalaureusetöö hõlmab predikaatarvutust toetavate veebitööriistadega tutvumist, et välja selgitada, milles predikaatarvutuse lahendamiskeskkonna kujundamisel sarnastelt programmidelt eeskuju võtta ja mida vältida. Käesolevas peatükis antakse lühiülevaade vaid mõningatest veebis leiduvatest tööriistadest.

10.1 A Predicate Logic Calculator

Antud veebirakendus [2] võimaldab kindlaks teha, kas kasutaja sisestatud lausearvutuse või ühekohaline predikaatarvutuse valem on samaselt tõene, samaselt väär või kehtestav. Lisaks avaldise väärtustamisele lubab programm teha järgnevaid operatsioone:

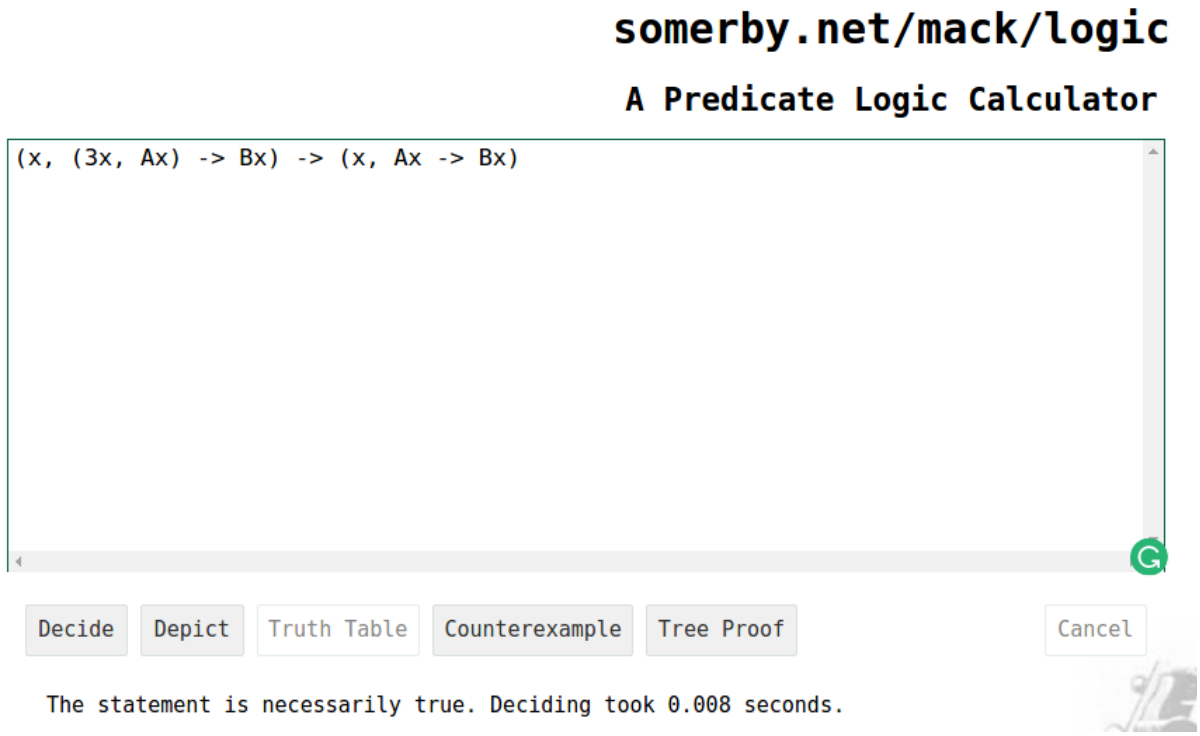
- kujutada graafiliselt väärtustamise aluseks olevat puud
- kirjeldada lausearvutuse valemi tõeväärtustabelit
- samaselt väära või kehtestava avaldise korral leida näite, mil avaldis väärtustub valeks
- navigeerida programmi Tree Proof Generator veebilehele [7], mis kujutab samaselt tõese avaldise tõesuspuud

Veebirakendus paistab silma põhjaliku dokumentatsiooni ja mitmete avaldise süntaksit puudutavate võimalustega, muuhulgas võimalus:

- kirjutada parseri poolt eiratavaid kommentaare
- defineerida oma sümboleid ning anda predikaatidele ja muutujatele tähenduslikke nimesid, asendades näiteks implikatsiooni väljendava märgi -> sõnaga „siis”
- kirjutada pikka avaldist mitmele reale
- kasutada kahte predikaati väljendavaid eeldefineeritud lühisümboleid, näiteks „SiP” väljendab seost „leidub element, mis on nii omadus S kui ka omadus P”.

Kuigi kõnealune veebirakendus sisaldab arendatava programmi seisukohast mitmeid kasulikke funktsioone, on selle üheks puuduseks tõsiasi, et rakendus ei toeta enam kui ühekohalisi predikaate ega avaldisi, mis sisaldavad vabu indiviidmuutujaid.

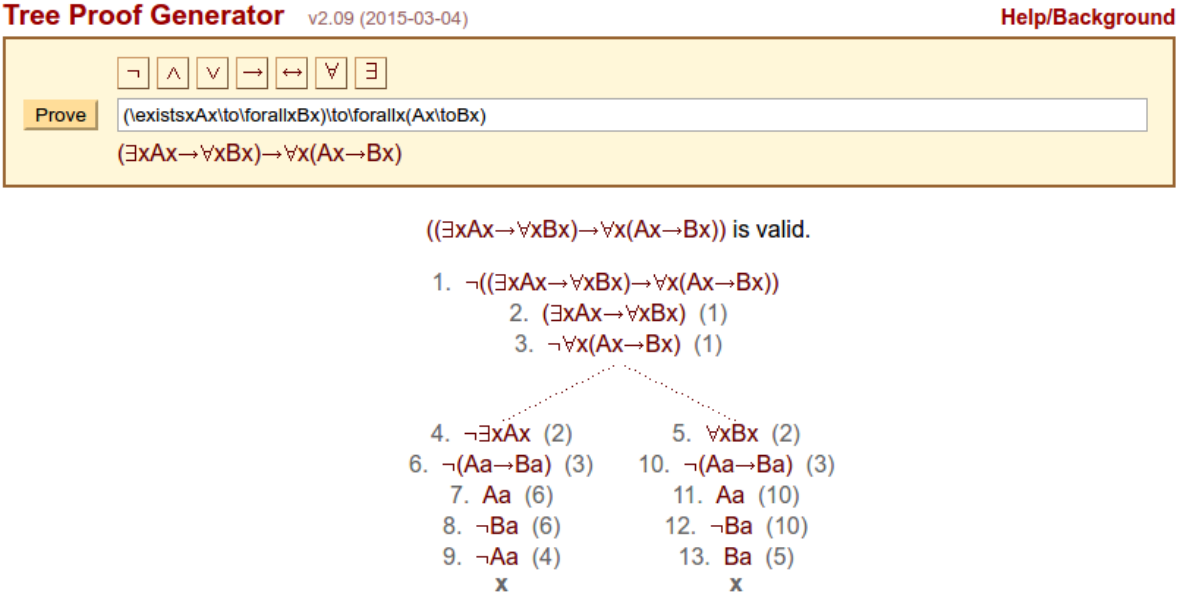
Samuti leiab käesoleva töö autor, et viis, kuidas analüüsitavas veebirakenduses tähistatakse kvantori rakendumise piire avaldise teatud osale, ei ole piisavalt intuitiivne. Selleks, et predikaatarvutusega algust tegev tudeng saaks keskenduda ülesande sisulisele poolele, taotleb arendatav programm vähem töömahukat ja ühtselt tõlgendatavat sulukasutust.



10.2 Tree Proof Generator

Antud veebiprogramm [7] kontrollib, kas sisestatud lausearvutuse või predikaatarvutuse valem on samaselt tõene või mitte. Samaselt tõese valemi jaoks koostab programm tõesuspuu. Valemi jaoks, mis ei ole samaselt tõene, toob programm näite interpretatsioonist, kus valem on väär.

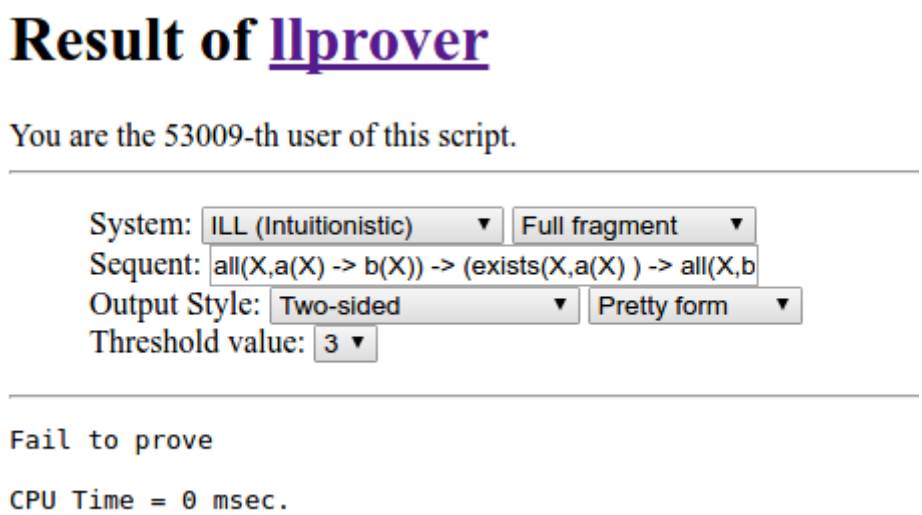
Programm on oma olemuselt lihtne, kuid sisaldab funktsionaalsust, mis kattub arendatava lahenduskeskkonna eesmärgiga, nagu tõesuspuu ja kontranäite kujutamine ning mitmekohaliste ja vabu muutujaid sisaldavate predikaatide toetamine. Samuti peab töö autor selle eeliseks paindlikku kasutajaliidest – valemit on vastavalt eelistusele võimalik sisestada, klikkides tehteid ja kvantoreid kujutavatele graafilistele nuppudele või trükkides tekstimärgendeid, näiteks `\neg`, `\forall`. Valemi kirjapilt on jooksvalt sisendi all nähtav.



10.3 A Linear Logic Prover

Antud veebirakendus [1] kasutab kasutaja valitud loogika tuletusreegleid, et teha kindlaks, kas sisestatud valem on samaselt tõene või mitte. Samaselt tõese valemi puhul kuvatakse konstrueeritud tuletuspuu, vastasel juhul antakse teada, et tõestamine ebaõnnestus.

Rakendus ei võimalda verifitseerida enam kui ühekohalise predikaatarvutuse valemi samaselt tõesust. Samuti ei paku rakendus eeskuju kasutusmugavuse seisukohast: oodatavat süntaksit tuleb otsida dokumentatsioonist, ning süntaksivea teade on vähe informatiivne.



Joonis 3

11 Kokkuvõte

Käesoleva bakalaureusetöö raames luuakse Java teek predikaatarvutuse väljendamisülesannete lahenduste kontrollimiseks. Vajadus taolise programmi järele on esile kerkinud Tartu Ülikooli matemaatilise loogika kursustel, kus harjutatakse predikaatarvutuse väljendamisülesandeid. Loodava teegi ja teeki sisaldava veebiprogrammi eesmärk on tõhustada nimetatud kursuste predikaatarvutuse osa praegust töökorraldust, kus praktikumijuhendajad kontrollivad tudengite väljendamisülesannete lahendusi käsitsi. Käsitsi kontrollimine on aga aeganõudev ja töömahukas. Kuna kahe predikaatarvutuse valemi samaväärsuse kontrollimine on algoritmiliselt mittelahenduv, ei asenda programm täielikult inimkontrollijat. Sellest hoolimata annab lahenduste kontrollimise automatiseerimine võimaluse harjutada teemat, mis valmistab paljudele tudengitele raskusi, ja osutada levinud veakohtadele, nagu väljendamisülesannete signatuuri mittejärgimine ja väär vabade muutujate kasutamine.

Kasutades tõesuspuu meetodit ja väärtuste väljaarvutamise meetodit, vastab loodud programm küsimusele, kas tudengi valem on samaväärne õige valemiga, ühel järgnevatest viisidest – jah, ei, ei tea. Need kaks meetodit on valitud põhjustel, et tõesuspuu meetod võimaldab kindlalt näidata kahe valemi samaväärsust, väärtuste väljaarvutamine aga kahe valemi mittesamaväärsust. Ebamäärane vastus (ei tea) on aga tingitud järgnevatest asjaoludest – tõesuspuu meetod ei tuvastanud, et kaks valemit on samaväärsed, kuna meetod piirdub valemi loogilise struktuuri analüüsimisega ega uuri aritmeetikatermide struktuuri, või tõesuspuu moodustamisel toimus valemi kujust tingituna tõesuspuu tippude arvu eksponentsiaalne kasv, mistõttu meetod ei jõudnud kindla vastuseni automaatse kontrollimise seisukohast otstarbeka aja jooksul. Kasutusele võeti väärtuste väljaarvutamise meetod, mis lõplikul naturaalarvude hulgal väärtustatuna ei leidnud väärtust, mil kahe valemi tõeväärtused on erinevad. See on olukord, mil vastus küsimusele, kas kaks valemit on samaväärsed, jääb lahtiseks.

11.1 Edasiarendamise võimalused

Võttes arvesse teegi implementeerimisel tehtud kitsendusi ja testimise tulemusi, on võimalik teeki edasi arendada ja täiendada järgnevalt:

- kontrollida muudes signatuurides – näiteks hulgateooria signatuuris – väljendatud valemite samaväärsust;
- täiendada tõesuspuu meetodit nii, et see arvestataks aritmeetikatermide struktuuri. Uurida termide võrdsust, korrutades lahti sulud ja järjestades muutujad tähestikuliselt. See aitaks tuvastada valemeid, mis on samaväärsed, kuid erinevad aritmeetikatermide struktuuri poolest.
- uurida tudengite vastuseid ja põhjusi, miks tõesuspuu meetod tudengite sisenditega tulemust ei anna. Täiendada tõesuspuu meetodit nii, et võimalikult väike osa tudengite sisenditest annaks vastuseks „ei tea”.

12 Lisad

12.1 Grammatika fail

```
grammar Pred;
koguvalem
    : abidef* predvalem;
abidef
    : predtahis ':=' predvalem
    ;
predtahis
    : predsymbol defargumendid*
    ;
predsymbol
    : identifikaator
    ;
identifikaator
    : TAHT (TAHT |ARV)*
    ;
defargumendid
    : '(' indiviidmuutuja (',' indiviidmuutuja)* ')'
    ;
indiviidmuutuja
    : TAHT
    ;
predvalem
    : ekvvalem ('~' ekvvalem)*
    ;
ekvvalem
    : implvalem ('->' implvalem)*
    ;
implvalem
    : disjvalem ('v' disjvalem)*
    ;
disjvalem
    : konjvalem ('&' konjvalem)*
    ;
```

```
konjvalem
    : ('¬' | iga | eks)* korgvalem
    ;
iga
    : '∀' indiviidmuutuja
    ;
eks
    : '∃' indiviidmuutuja
    ;
korgvalem
    : '(' predvalem ')' | atomaarnevalem
    ;
atomaarnevalem
   : predsymbol termargumendid*
   | term '=' term
   ;
termargumendid
    : '(' term (',' term)* ')'
     ;
term
    : pmterm ('+' pmterm)*
    ;
pmterm
    : kjterm ('*' kjterm)*
    ;
kjterm
    : '0' | '1' | indiviidmuutuja | '(' term ')'
    ;
ARV
    : [0-9];
TAHT
    : [a-zA-Z]
    ;
WS: [ \t\r\n]+ -> skip;
```

13 Viidatud kirjandus

[1] A Linear Logic Prover. http://bach.istc.kobe-u.ac.jp/llprover/ (10.01.2017)

[2] A Predicate Logic Calculator. http://somerby.net/mack/logic/index.html (10.01.2017)

[3] Palm R. Diskreetse matemaatika elemendid. Tartu: Tartu Ülikooli Kirjastus. 2009.

[4] Palm R., Prank R. Sissejuhatus matemaatilisse loogikasse. Tartu: Tartu Ülikooli Kirjastus. 2004.

[5] Parr, T. The Definitive ANTLR 4 Reference. United States of America. 2012.

[6] Prank R. Predikaadid ja kvantorid. Esimest järku keeled. Loengukonspekt. https://moodle.ut.ee/mod/resource/view.php?id=229848 (9.08.2017)

[7] Tree Proof Generator. http://www.umsu.de/logik/trees/ (10.01.2017)

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Siiri Saar**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   Teek predikaatarvutuse väljendamisülesannete lahenduste kontrollimiseks,
   (*lõputöö pealkiri*)

   mille juhendaja on Reimo Palm,
   (*juhendaja nimi*)

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.08.2017**