VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Metamath kalba ir jos vizualizavimas

(Metamath Language And Its Visualization)

Atliko: IV kurso I grupės studentė
Rasa Urbonaitė (parašas)

Darbo vadovas:
lekt. Irus Grinis (parašas)

Vilnius 2025

Turinys

Summary

The objective of this study is to examine the Metamath programming language and its role in generating or validating mathematical proofs. This paper proposes improvements to the visualisation techniques to enhance their accessibility for individuals with a mathematical background. Additionally, it addresses the limitations of Metamath in terms of visualisation. It also examines the potential difficulties users may encounter in understanding the language and its applications. This study examines the language aspect of Metamath, focusing on the programming languages „Python“ and „Brython“. This tool can assist in the learning and improvement of mathematical understanding. The objective is to develop a visualisation tool to facilitate the understanding of the Metamath language. The text examines the use of the language for the creation and demonstration of mathematical statements, taking into account the visualisation challenges faced by Metamath. The study is aimed at users' understanding of language use and appropriate application. The objective of this study is to identify methods for visually representing the content of Metamath in a manner that is comprehensible to non–mathematicians. By examining the potential programming languages „Python“ and „Brython“, it is possible to create a supporting web page. The study will result in the development of a visualisation tool to facilitate the interpretation of Metamath language texts. Abbreviations and technical terms will be explained at first use. Furthermore, the potential for enhancing the utilisation of the Metamath language with existing tools will be evaluated. This will facilitate the development of new tools, thereby facilitating the comprehension of mathematical concepts.

Santrauka

Šio tyrimo tikslas – ištirti „Metamath“ programavimo kalbą ir jos vaidmenį generuojant ar patvirtinant matematinius įrodymus. Šiame dokumente siūlomi būdai, kaip pagerinti vizualizavimo metodus, kad matematikos žinių turintiems žmonėms būtų lengviau suprasti. Be to, jame nagrinėjami „Metamath“ apribojimai, kai kalbama apie vizualizavimą. Kaip vartotojas gali susidurti su sunkumais suvokdamas kalbą ir jos taikomąsias programas. Šiame tyrime nagrinėjamas „Metamath“ kalbos aspektas, daugiausia dėmesio skiriant programavimo kalboms „Python“ ir „Brython“. Šis įrankis gali padėti mokytis ir pagerinti matematinį supratimą. Tikslas yra sukurti vizualizavimo įrankį, kuris palengvintų „Metamath“ kalbos supratimą. Tekste nagrinėjamas kalbos naudojimas kuriant ir demonstruojant matematinius teiginius, atsižvelgiant į vizualizavimo iššūkius, su kuriais susiduria „Metamath“. Tyrimas skirtas vartotojams suprasti kalbos naudojimą ir tinkamą taikymą. Šio tyrimo tikslas – nustatyti metodus, kaip vizualiai pavaizduoti „Metamath“ turinį suprantamu būdu ne matematikams. Išnagrinėjus galimus „Python“ ir „Brython“ programavimo kalbas, kaip galima sukurti pagalbini tinklapį. Tyrimas padės sukurti vizualizavimo įrankį, palengvinantį „Metamath“ kalbos tekstų interpretaciją. Santraukos ir techniniai terminai bus paaiškinti pirmą kartą panaudojus. Be to, įvertinsime galimybes pagerinti „Metamath“ kalbos naudojimą turimais įrankiais. Tai palengvins naujų priemonių kūrimą, o tai leis lengviau suprasti matematines sąvokas.

Raktiniai žodžiai: Metamath kalba, vizualizacija, Metamath specifikacija

Įvadas

Matematika, atsiradusi nuo antikos laikų[Nor23a], yra esminis pažinimo įrankis, padedantis tyrinėti ir suprasti pasaulį. Kartu su matematika išsivystė matematinis įrodymas, kuris yra būdas patvirtinti tam tikrą matematikos teiginį, laikantis priimtų srities aksiomų. Pavyzdžiui, vienas iš žinomiausių matematinių įrodymų yra Pitagoro teorema, kurią pirmą kartą iškėlė Pitagoras Graikijoje apie 2000–1500 metais prieš Kristų[Nor23b]. Matematiniai įrodymai dažniausiai yra sudėtingi ir ilgi procesai, kurie gali būti padaryti iš kitų įrodymų. Todėl juos kuriant lengva padaryti klaidas. Tam, kad būtų išvengta klaidų ir užtikrinto matematinio teiginio teisingumo, yra naudojamos įvairios sistemos, kurios patikrina matematinį įrodymą. Jei klaida pastebima, ji gali būti ištaisyta. Viena iš šių sistemų yra programinė kalba „Metamath“. Šio darbo tikslas yra sukurti būdą vizualizuoti „Metamath“ failus, kurie padėtų suprasti naudotojo pateiktus įrodymus, palengvintų vartotojo sąveiką su sistema ir suteiktų galimybę dalintis išvadomis su kitais tyrėjais.

Darbo tikslas

Šio darbo tikslas yra vizualizuoti „Metamath“ taip, kad vartotojui būtų lengviau naudotis šia kalba ir galėtų analizuoti rezultatus.

Uždaviniai

- Išanalizuoti „*Metamath*“ kalbos struktūrą ir veikimo principus.
- Susipažinti su esamais Metamath įrankiais ir jų galimybėmis.
- Įvertinti vizualizacijos technikas, tinkamas formalios logikos sistemoms.
- Parinkti tinkamas technologijas („*Python*“, „*Brython*“, „*SVG*“) interaktyviam įrankiui realizuoti.
- Sukurti programinį sprendimą, leidžiantį vartotojui vizualizuoti Metamath „.mm“ failų turinį.
- Įdiegti pagrindines funkcijas: įkėlimą, analizę, žingsnių vizualizavimą, komentarų pateikimą.
- Įvertinti sukurtos sistemos veikimą ir pateikti galimus tobulinimo variantus.

Darbo aktualumas

Formalių įrodymų vizualizacija vis dar išlieka ribotai išvystyta sritis. Nors „Metamath“ kalba leidžia tiksliai aprašyti matematinius įrodymus, jos formali sintaksė dažnai apsunkina supratimą vartotojams, net ir turintiems matematinį pagrindą. Šis darbas siekia sumažinti šią atskirtį, pasiūlant priemonę, leidžiančią lengviau suprasti įrodymų logiką per grafinius atvaizdus.

1. Metamath

„Metamath“ sukūrė Normanas Megelis 1990–ųjų pabaigoje. su konkrečiu tikslu – sukurti matematikos formalizavimo sistemą, kuri būtų griežta, aiški ir vienareikšmiška, pabrėžiant matematikos struktūrą, o ne jos turinį. Ši sistema teikia pirmenybę neformaliems matematiniams samprotavimams, suteikdama jų pagrindu remiamą formalų kalbos standartą ir įrodymų tikrinimo mechanizmą. „Metamath“ pirmą kartą buvo pritaikyta matematikos srityje, kur ji įgalino kurti ir tikrinti matematinius įrodymus. Tai papildė struktūrą aksiomoms, apibrėžimais ir teorijomis, leisdama vartotojui sistemingai sudaryti ir patikrinti matematinius įrodymus žingsnis po žingsnio. Šios sistemos egzistavimas padėjo užtikrinti matematinių sampratų teisingumą, todėl ji tapo nepakeičiama priemone moksliniams tyrimams ir švietimui. Viena iš svarbiausių „Metamath“ savybių yra plati teoremų biblioteka. Šioje bibliotekoje yra tūkstančiai teoremų ir įrodymų, apimančių įvairias matematikos sritis. Biblioteka nuolat plečiama, nes bendruomenė nuolat prideda naujų teoremų. Svarbu pabrėžti tam tikras teoremas kaip svarbiausias, nes jų reikšmė gali kisti priklausomai nuo konteksto ir taikymo srities.

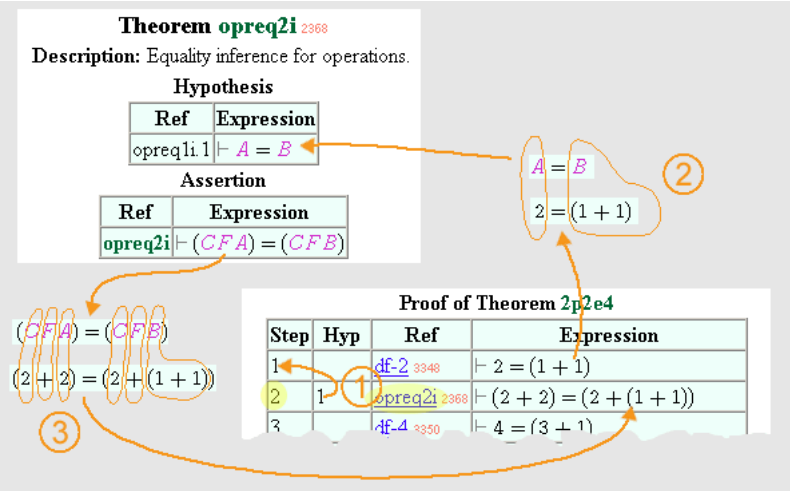
1.1. Kaip Metamath įrodymai veikia

Vokietijos matematikas Davidas Hilbertas yra pasakęs: „Teorema matematikos yra nepilna, kol ji nesuprantama taip gerai, kad galėtum paaiškinti ją pirmam žmogui, kurį sutinki gatvėje.“ Tai taip pat galioja ir „Metamath“, nes norint naudotis šia kalba, reikia bent šiek tiek suprasti jos veikimą. Pradėkime nuo to, kaip paaiškinamas „Metamath“ veikimas. Tam, kad suprastumėte, kaip tai veikia, pradžioje reikėtų suprasti, kaip veikia simbolių rinkinys . Tai yra būtinas žingsnis, norint tapti „Metamath“ naudotoju.

„Metamath“ yra konceptualiai labai paprasta sistema, bet tai turi savo kainą. Dažnai reikia atlikti daugybę žingsnių, kad galėtumėte visiškai įrodyti kažką, tačiau šie įrodymai yra tikrinami kompiuteriu, kad galite pasitikėti jų tikslumu. Norint suprasti, kaip vyksta simbolių manipuliacija „Metamath“ įrodymuose, reikia suprasti sąvoką „pakeitimas“. Tai apima simbolių keitimą naudojant išraiškas, kurios yra taikomos specialiems kintamųjų atvejams. Pavyzdžiui, mokykloje mes išmokome, kad a + b = b + a, kur a ir b yra kintamieji. Du pakeitimo atvejai šiuo principu yra $5 + 3 = 3 + 5$ ir $(x - 7) + c = c + (x–7)$.

Tai yra vienintelė matematikos sąvoka, kurią reikia žinoti naudojant „Metamath“. Jos veikimas bus parodytas žingsnių įrodyme. Pavyzdžiui, galime analizuoti įrodymą 2 + 2 = 4, žingsnius. Tai bus pateikta naudojant šiuos pavyzdžius. Jums nereikia suprasti simbolinę reikšmę; pakanka žinoti pateiktas taisykles, kaip manipuliuoti jomis norint įrodyti teoremą. Palyginus tai su ilgalaikėmis studijomis, kurių dažnai reikia, siekiant giliau suprasti ir patvirtinti įrodymus, „Metamath“ gali pasirodyti ribotas. Nors ji negali padėti su visais ilgaisiais žingsniais, kuriuos dažnai mes žmogiškai atliekame intuityviai, ji yra nepakeičiama, jei norime užtikrinti, kad matematikos simbolių eilė, vadinama „teorema“, yra mechaninė išvada iš aksiomų rezultatų. Tai padeda pasiekti aiškias galimybes įrodyti, taip įsisavinti tam tikras matematines savybes.[Meg05].

„Metamath“ koncepcinis paprastumas turi savo trūkumą : dažnai reikalingas didelis žingsnių skaičius, norint gauti pilną įrodymą nuo aksiomų iki galutinių išvadų. Tačiau įrodymai yra patvirtinti kompiuteriu, ir galite pasirinkti tik tų žingsnių studijavimą, kurie jus dominą, ir vis tiek turėti visišką tikrumą, kad kiti žingsniai yra teisingi.



1 pav. 2p2e4 įrodymo antras žingsnis remiasi pirmuoju žingsniu, kuris savo ruožtu „patalpina“ anksčiau teoremos oveq2i hipotezę (kuri buvo vadinta opreq2i). Oveq2i išvada (teiginys) tada generuoja 2p2e4 antrą žingsnį. Atidžiai atkreipkite dėmesį į pakeitimus (pasirinktus plonais oranžiniais linijomis), kurie vyksta.

1 pav. matome 2 + 2 = 4 teoremos įrodomą dalį ,vadintą 2p2e4, duomenų bazėje. Parodysime, kaip pasiekėm 2–ojo žingsnio įrodymą, kuriame teigiama, kad (2 + 2) = (2 + (1 + 1)).

1. Nagrinėjant antrojo žingsnio įrodymą Ref stulpelyje matosi, kad jis remiasi anksčiau įrodyta teorema, vadinama oveq2i. Teorema oveq2i turi prielaidą, o 2–ojo žingsnio Hyp stulpelyje nurodome, kad 1–as žingsnis patenkins šią prielaidą.

2. Padarome pakeitimus prielaidos kintamiesiems, kad jie atitiktų simbolių eilutę 1–ojo žingsnio Išraiškos stulpelyje. Tam pakeičiame išraišką „2“ kintamajam **A** ir išraišką „(1 + 1)“ kintamajam B. Oveq2i prielaidos vidurinis simbolis yra „=“, kuris yra konstanta, ir mes negalime pakeisti konstantos. Konstantos turi tiksliai sutapti.
   Tai lengvai atpažįstama. Mūsų pavyzdyje violetinės didžiosios pasvirusios raidės yra kintamieji, o simboliai „(“, „)“, „1“, „2“, „=“, ir „+“ yra konstantos.[Meg21];

   Šiame pavyzdyje tikrinama ar konstantos simboliai žinomi. Kitais atvejais jų gali ir nebūti. Reikia atkreipti dėmesį, ar simboliai yra kintamieji, ar konstantos, o ne į tai, ką jie „reiškia“. Tikslas yra nustatyti, kokius keitimus į nurodytos teoremos kintamuosius reikia atlikti, kad simbolių eilutė atitiktų.

3. Oveq2i Teiginio langelio išraiškos stulpelyje yra keturi kintamieji: A, B, C ir F. Kadangi jau atlikti pakeitimai prielaidose, kintamiesiems A ir B priskirtos atitinkamos reikšmės „2“ ir

„(1 + 1)“, ir šių priskyrimų negalima keisti. Tačiau naujiems kintamiesiems C ir F galima priskirti bet kokias reiškinio išraiškas. Pakeisdami „2“ į C ir „+“ į F gauname (2 + 2) = (2 + (1 + 1)), kurią rodo išraiškos stulpelyje 2–ąjį 2p2e4 įrodymo žingsnį.

Pastebėjimai dėl pakeitimų:

- Pakeitimai „Metamath“ yra atliekami vienu metu, tai reiškia, kad kiekvienas kintamojo veiksmas turi būti pakeistas ta pačia išraiška, kuri yra nurodyta teoremoje.
- Pakeitimai yra atliekami tik nurodytos teoremos kintamuosiuose, o ne kokio nors įrodymo žingsnyje, kuris yra nurodytas „Hyp“ stulpelyje, kintamuosiuose.
- Atliekant pakeitimus, būtina laikytis taisyklingos sintaksės reikalavimų, pavyzdžiui, simbolių eilutė turi atitikti klasės išraišką, kad būtų išvengta dviprasmybių.

2. Metamath elementai ir jų semantinė reikšmė

Formalaus matematinio pagrindimo sistemose, tokiose kaip „Metamath“, svarbu turėti aiškią, kompaktišką ir vienareikšmiškai interpretuojamą kalbą. Ši kalba leidžia ne tik apibrėžti aksiomas ir taisykles, bet ir formaliai konstruoti bei tikrinti matematinius įrodymus. Norint efektyviai dirbti su Metamath sistema, būtina suprasti jos kalbos elementus, jų sintaksinę formą ir semantinę funkciją. Žemiau pateikiami pagrindiniai Metamath kalbos komponentai:

- **Komentarai** – prasideda **„$(”** ir baigiasi **„$)”** simboliais. Juose parašomas komentaras.
- **Konstantų apibrėžimas** – prasideda **„$c”** ir baigiamas **„$.“**. Juose aprašomos konstantos, kurių negalima keisti.
- **Kintamųjų apibrėžimas** – prasideda **„$v”** simboliais ir baigiasi **„$.“**. Juose aprašomi kintamieji, kuriuos galima keisti sintaksės metu.
- **Kintamųjų apribojimas** – nurodant anksčiau apibrėžtus kintamuosius tarp **„$d“** ir **„$.”** simbolių sekas. Šis apribojimas padeda nustatyti ir manipuliuoti kintamaisiais naudojant pakeitimo taisykles.
- **Apibrėžimas hipotezės** – prasideda **„$e“** ir baigiasi **„$.“**. Jo viduje rašomos seka hipotezėje matematinių simbolių, kurią sudaro konstantos ar kintamieji, išreiškia loginę tiesą.
- **Naudojamos hipotezės** – parašoma naudojantis **„$f”** simboliais ir baigiama **„$.“**. Norint juo naudotis, turi būti jau paminėta iš anksto.
- **Blokai grupavimas** – parašoma naudojantis **„${”** simboliais ir baigiama **„$}“**. Suriša hipotezes su aksiomomis ir teoremomis.
- **Failo įtraukimas** – parašoma naudojantis **„$[”** simboliais ir baigiama **„$]“**. Įtraukia failą kuris yra užrašytas viduje.

2.1. Komentarai „$(” ir „$)”

„*Metamath*“ kalboje komentarai žymimi simbolių pora **„$(”** ir baigiasi **„$)”**. Tarp šių simbolių įrašytas tekstas nėra interpretuojamas tikrintuvo ir naudojamas tik dokumentacijai ar žmogaus supratimui. Komentarai gali būti naudojami bet kurioje Metamath failo vietoje, padedant aiškinti įrodymus ar struktūrą. Pavyzdys kaip gali atrodyti kode komentarai.

```
($ komentras virš bloko $)
${ ($ gali būti bloke $)
1a $a ⊢ 2  $( Gali buti pačiame įrodyme parašytas )$ $.
$}
```

2.2. Konstantos

Konstantos apibrėžiamos naudojant „$c ... $.“ sintaksę. Tai yra simboliai, kurie žymi nekintamus objektus: skaičius, loginės algebros ženklus ar kitus formalius operatorius. Konstantos apibrėžiamos pradžioje ir yra visos sistemos pagrindas – jų reikšmės negali būti keičiamos.

```
$c 0 1 2 3 = + wff ( ) $.
```

Ši eilutė deklaruoja pagrindines simbolines konstantas, kurios bus naudojamos tolimesniuose apibrėžimuose ar įrodymuose.

2.3. Kintamųjų apibrėžimas

Kintamieji deklaruojami naudojant „$v ... $.“ formą. Jie skirti universaliems simboliams, kurie gali būti keičiami ar instancijuojami į konkrečius objektus formulėse. Šie kintamieji leidžia kurti bendresnes taisykles bei šablonus, taikomus įvairiems atvejams.

```
$v x y z $.
```

Šiame pavyzdyje deklaruojami trys kintamieji, kurie gali būti naudojami formulėse ar taisyklėse.

2.4. Kintamųjų apribojimas (disjunkcija)

Naudojant „$d ... $.“, galima nurodyti, kad tam tikri kintamieji turi būti interpretuojami kaip nepriklausomi vienas nuo kito. Tokie apribojimai yra svarbūs, kai siekiama išvengti neteisėtų kintamųjų sustumtumų įrodymo metu, ypač kai taikomos pakaitos taisyklės.

```
$d x y $.
```

Reiškia, kad x ir y turi būti laikomi skirtingais kintamaisiais.

2.5. Hipotezių apibrėžimas

Hipotezės Metamath kalboje žymimos simboliu „$e“ ir baigiamas „$.“. Tai išraiškos, kurias laikome prielaidomis tam tikros taisyklės ar teoremos kontekste. Hipotezės sudaro įrodymo loginį pagrindą.

```
h1 $e ⊢ x ∈ ℝ $.
```

Ši hipotezė sako, kad x priklauso realiųjų skaičių aibei.

2.6. Fiksuotų formulių priskyrimas

Naudojant „$f ... $.“ žymėjimą, kiekvienas simbolis ar formulė susiejama su konkrečia sintaksine kategorija (pavyzdžiui, kad x yra aibės kintamasis, o φ – formulė). Tokie susiejimai leidžia tiksliai apibrėžti formulės tipą ir užtikrinti teisingą jos vartojimą įrodymuose.

Šio tipo įrašas visada turi griežtą struktūrą:

```
etiketė $f tipas kintamasis $.
```

Čia:

- **etiketė** – identifikatorius, žymintis šį apibrėžimą. Nors neprivalomas, dažnai naudojamas aiškumui užtikrinti.
- **$f** – nurodo, kad tai yra vadinamoji „floating hypothesis“ – tipas, priskirtas kintamajam.
- **tipas** – sintaksinė kategorija, kuriai priklauso kintamasis. Dažniausiai naudojami tipai:
  - `wff` – „well-formed formula“ (gerai suformuota formulė). Pvz., šis įrašas sako, kad φ yra wff:

    ```
    wph $f wff φ $.
    ```

  - `class` – klasės išraiška. Pvz., simbolis A priklauso klasei:

    ```
    clA $f class A $.
    ```

  - `set` – aibės kintamasis. Pvz.:

    ```
    set1 $f set x $.
    ```

  - $\vdash$ – įrodomas teiginys (provable statement). Tai reiškia, kad φ yra teiginys, kurį siekiama įrodyti:

    ```
    h1 $f ⊢ φ $.
    ```

  - `term` – terminas. Naudojamas, kai aprašomi bendresni matematiniai terminai:

    ```
    t $f term t $.
    ```

  - `nat` – natūralusis skaičius. Naudojamas dirbant su skaitinėmis reikšmėmis:

    ```
    n $f nat n $.
    ```

- **kintamasis** – pats simbolis, kuris vėliau bus naudojamas formulėse ir įrodymuose.

2.7. Aksiomos

Aksiomos ir apibrėžimai žymimi naudojant „$a“ ir baigiamas „$.“ Aksiomos laikomos pradinėmis tiesomis, iš kurių konstruojamos likusios išvados. Ši žymėjimo forma nurodo, kad pateikiama formulė yra laikoma pagrindine prielaida arba apibrėžimu, kuris nereikalauja įrodymo. Jis yra prijamamas be įrodymų. Aksiomos sistemoje veikia kaip pradiniai statybiniai blokai – tai formalios taisyklės ar teiginiai, kurie laikomi iš anksto priimtomis tiesomis, nepriklausomai nuo jų semantinio pagrindimo. Aksiomos gali apibrėžti:

- Aritmetines tapatybes (pvz., „1 + 1 = 2“),
- Loginės struktūros taisykles (pvz., implikaciją ar ekvivalentiškumą),
- Struktūrinius pagrindus, tokius kaip elementų priklausomybes, priklausomybės taisykles ir taip toliau.

```
ax-id $a ⊢ ( x = x ) $.
```

2.8. Teoremos

Teoremos Metamath kalboje žymimos simboliu „$p“ ir užbaigiamos „$.“. Skirtingai nuo aksiomų („$a“), kurios yra priimamos be įrodymo, teoremos privalo turėti pagrindimą – tai yra, jos turi būti įrodomos remiantis anksčiau priimtomis aksiomomis, apibrėžimais ar jau įrodytomis kitomis teoremomis. Svarbi teoremos dalis yra įrodymo seka, kuri aprašoma po simbolio „$=“. Šioje sekcijoje nurodoma, kokie anksčiau žinomi ar įrodyti rezultatai buvo panaudoti pagrindinio teiginio išvedimui. Ši seka iš esmės atitinka įrodymo žingsnius, kurie rodo, kaip loginė grandinė veda nuo hipotezių prie išvados. Taipogi, reikia pasirūpinti, kad kiekvienas įrodymo žingsnis turi remtis tik iš anksto apibrėžtomis loginėmis taisyklėmis.

```
wnew $p wff ( s -> ( r -> p ) ) $= ws wr wp w2 w2 $.
```

Šioje teoremoje įrodomą, kad „$(s- > (r- > p))$“ naudojant iš anksčiau deklaruotų konstantų, hipotezių aksiomų.

2.9. Blokų grupavimas

Struktūriniam kodo organizavimui naudojamos grupės žymimos simbolių pora „${“ ir „$}“. Tokie blokai padeda logiškai atskirti susijusias deklaracijas, ypač kai įrodymų ar apibrėžimų failas tampa ilgas ir sudėtingas. Taip leidžia surišti hipotezes su taisyklėmis (aksiomomis ar teoremomis) ir valdyti kintamųjų galiojimo sritis.

- Toks grupavimas leidžia atskirti teiginių sritis, kad viena aksioma ar teorema nepaveiktų kitos.

- Leidžia aprašyti teoremą ar aksiomą su hipotezėmis (naudodamas „$e“), tos hipotezės galioja tik tame pačiame bloke su „$a“ ar „$p“.

```
${ $( 1 + 1 = 2 $)
  h1 $e ⊢ 1 $.
  h2 $e ⊢ 1 $.
  add1 $a ⊢ ( 1 + 1 ) = 2 $.
$}
```

- Padeda apriboti kintamuosius kuriai galioją tik bloko viduje. Šiuo atveju kintamųjų apribojimas $x \neq y$ galioja tik šiame bloke. Už bloko ribų šis „$d“ nebegalioja

```
${
  $d x y $.
  ax1 $a ⊢ ∀ x ∃ y ( P x y ) $.
$}
```

2.10. Kodo pavyzdys

Dabar parodysiu kodo pavyzdį kuriame yra naudojami: konstantos, kintamieji, apibrėžimo hipotezės, naudojamos hipotezės, blokų grandinės ir įrodymai. Čia bus įrodoma, kad $\vdash -t = t$.

1. Pirmiausia reikia deklaruoti konstantų ir kintamuosius

```
1  $c 0 + = → ( ) term wff ⊢ $.
2  $v t r s P Q $.
```

   - $c — konstantos: 0, +, =, →, (,), term, wff, .
   - $v — metavariablės: t, r, s (terminai), P, Q (wff).

2. Fiksuotų formulių priskyrimas:

```
3  tt $f term t $.
4  tr $f term r $.
5  ts $f term s $.
6  wp $f wff P $.
7  wq $f wff Q $.
```

   - $f reiškia, kad metavariablė turi tam tikrą tipą.
   - „tt“ reiškia, kad t yra terminas.
   - „wp“ reiškia, kad P yra wff (teiginys).

3. Termino apibrėžimai:

```
8  tze $a term 0 $.
9  tpl $a term ( t + r ) $.
```

   - $a reiškia aksiomą ar taisyklę.
   - tze: 0 yra terminas.
   - tpl: (t + r) yra terminas.

4. Wff (teiginio) apibrėžimai:

```
10 weq $a wff t = r $.
11 wim $a wff ( P -> Q ) $.
```

   - weq: t = r yra wff.
   - wim: (P -> Q) yra wff.

5. Aksiomos:

```
12 a1 $a |- ( t = r -> ( t = s -> r = s ) ) $.
13 a2 $a |- ( t + 0 ) = t $.
```

   - a1 — tranzityvumo aksioma lygybei.
   - a2 — neutralumo (pridėjus 0) aksioma.

6. Modus ponens taisyklė:

```
14 ${
15   min $e |- P $.
16   maj $e |- ( P -> Q ) $.
17   mp  $a |- Q $.
18 $}
```

7. Teoremos įrodymas:

```
19 th1 $p |- t = t $=
20 tt tze tpl tt weq tt tt weq tt a2 tt tze tpl
21 tt weq tt tze tpl tt weq tt tt weq wim tt a2
23 tt tze tpl tt tt a1 mp mp $.
```

3. Metamath kalbos įrankiai

„*Metamath*“ yra veikianti sistema, ir šiuo metu yra sukurti tam tikri įrankiai, kurie suteikia galimybę patikrinti, ar formulės ir įrodymai yra teisingi. Kadangi pati „*Metamath*“ yra minimalistinė sistema. Labiau orientuotą į bendra pagrindą, kad efektyviai ir lengviau naudoti ją reikėtų papildomų įrankių. Todėl yra kuriama ir sukurtą visa eilė programų ir bibliotekos, kurios palengvina įrodymų kūrimą, redagavimą, tikrinimą ir analizavimą. Šiame kontekste apžvelgsime kai kuriuos iš dažniau naudojamus įrankius:

- **metamath-lamp** – tai naujas įrodymų asistentas, kuris skirtas darbiti kur su „Metamath“ įrodymų formatų sistema. Jo kūrėjas yra Igoras Ieskovas. Šiuo metu jis vis dar ne visiškai išvystytas, tačiau jau prieinamas naudotojams. Jo tikslas yra palengvinti darbą su „Metamath“, nereikalaujant išsamios sistemos veikimo žinios. Šis įrankis leidžia tikrinti įrodymus ir sekti teorijų vystymąsi. [Ies22]

- **mmj2** – tai vienas iš senesniu ir labiausiai naudojamų „Metamath“ įrankių. Jis buvo sukurtas naudojant Javą, o jo autorius yra David A. Wheeler. Su šiuo įrankiu galima ne tik skaityti ir analizuoti funkcijas per komandinę eilutę, bet ir naudoti grafinę vartotojo sąsają. Pagrindinė funkcija padeda kurti ir tikrinti įrodymus, vizualiai rodyti jų struktūrą, bei atlikti semantinę analizę. Gali automatiškai patikrinti kintamųjų naudojimą, simbolių leidimą, hipotezių deringuma ir kitokia formulias. Taipogi, turi greita atsakomoji ryšį apie klaidas arba apžvalga, taip pateikdamas aiškų ir efektyvų atsakymą. Gali palaikyti darbą su didekėmis duomenų bazėmis, kaip pavyzdžiu su „set.mm“, todėl yra plačiau naudojamas bendruomenėje.[Car12]

- **yamma** – (Yet Another Metamath Mode for Authors) vienas iš naujesnių įrankių, orientuotas į vartotojų patogumą ir intergraciją su populeria programinia įranga. Pasižymi sintakses ryškinimų, greitų klaidų pranešimų rialiuojų laiku, automatiškų funkcijų užbaigimų. Taip prisidėdamas prie efektyvaus redagavimo proceso ir mažindamas tipines sintaksės arba semantikos klaidas. Taipogi ypač naudingas teims, kurie nori dirbti su „*Metamath*“ naudojantis „Visual Studio Code“ kuris yra vienas išpoliarsnių kodų redaktorių. [gla12]

- **mmverify** – tai „Python“ kalba sukurtas verifikacijos įrankis, skirtas automatiškai tikrinti „Metamath“ kalba parašytų įrodymų korektiškumą. Šis įrankis atlieka itin svarbų vaidmenį automatizuojant įrodymų tikrinimo procesą – be jo kiekvieną įrodymo žingsnį tektų vertinti rankiniu būdu.Naudodamas „mmverify“, vartotojas gali patikrinti, ar visi įrodymo žingsniai yra logiškai pagrįsti ir atitinka „Metamath“ taisykles. Įrankis geba aptikti įvairias klaidas, tokias kaip netinkamas kintamųjų naudojimas, nesuderinamos hipotezės ar neteisingai suformuluotos išvados. Taigi šis įrankis leidžia užtikrinti, kad kiekvienas įrodymas yra ne tik intuityviai ar žodine prasme teisingas, bet ir formaliai korektiškas pagal griežtus loginius reikalavimus.[Lev]

Žinoma, yra ir kitų sukurtų „Metamath“ įrankių, tačiau šiame kontekste paminėti keli iš žinomiausių ir daugiausiai naudojamų.

3.1. „mmverify“

„Metamath“ – tai formali sistema, skirta matematinių įrodymų pateikimui kompiuteriui suprantamu formatu. Metamath Proof Explorer (MPE) – tai „*Metamath*“ kalba užkoduotų matematinių įrodymų rinkinys. Viena iš svarbiausių šios sistemos priemonių – „mmverify“ – naudojama įrodymų teisingumui automatiškai tikrinti.„mmverify“ tikrina kiekvieną įrodymo žingsnį, vertindama jo loginį pagrįstumą pagal Metamath sistemoje apibrėžtas taisykles. Ji užtikrina, kad įrodymo žingsniai būtų nuoseklūs ir kad išvados logiškai sektų iš pateiktų prielaidų. Jei įrodymas atitinka visas taisykles, laikoma, kad teorema ar teiginys yra formaliai teisingas sistemoje, kurią apibrėžia „*Metamath*“.Šis įrankis yra itin vertingas siekiant užtikrinti įrodymų korektiškumą ir stiprinti pasitikėjimą formalių matematinių rezultatų tikslumu. Jis plačiai naudojamas formalaus tikrinimo (angl. formal verification) ir automatizuoto teoremų įrodymo srityse.

- **Loginis nuoseklumas**: kiekvienas įrodymo žingsnis turi būti logiškai išvedamas iš ankstesnių žingsnių ar hipotezių, laikantis taisyklių.
- **Taisyklių laikymasis**: visi naudojami simboliai turi atitikti „*Metamath*“ sintaksę, įskaitant kintamųjų naudojimą ir įrodymų struktūrą.
- **Hipotezių išvadų ryšys**: tikrinama, ar teoremos išvados logiškai seka iš hipotezių, remiantis pateiktu įrodymu.
- **Deklaracijų tikrinimas**: įsitikinama ar visi duoti simboliai yra tinkamai deklaruoti kaip konstantos, kintamieji, aksiomos ar teoremos. .

3.1.1. *MM* klasė

„MM“klasė („**MM**“ yra trumpinys iš Metamath) esanti faile mmverify.py, yra pagrindinis komponentas, atsakingas už „Metamath“ duomenų bazės analizę bei įrodymų tikrinimą. Ši klasė atlieka įvairias užduotis, užtikrinant, kad tiek duomenų bazė, tiek įrodymai būtų teisingi ir atitiktų „Metamath“ kalbos reikalavimus. Pagrindinės **„MM“** klasės atliekamos funkcijos:

- Duomenų bazės analizė ir nuskaitymas: klasė gali nuskaityti Metamath duomenų bazę, duotą „.mm“ failą ir interpretuoti ją pagal Metamath sintaksę. Tai apima kintamųjų, konstantų, hipotezių ir teoremų skaitymą, kad būtų galima tinkamai suprasti ir apdoroti duomenų bazę.
- Hipotezių ir teoremų tikrinimas: klasė tikrina hipotezes ir teoremas, kad įsitikintų, jog jos yra logiškai teisingos ir atitinka Metamath kalbos taisykles. Tai apima kintamųjų naudojimą, hipotezių rūšiavimą, teoremų įrodymo tikrinimą ir kt.

- Įrodymų tikrinimas: klasė gali patikrinti pateiktus įrodymus, kad įsitikintų, jog jie atitinka teoremą, kurią jie turi įrodyti. Tai viena iš svarbiausių funkcijų, nes užtikrina, kad teoremos būtų tinkamai patikrintos.

- Klaidų aptikimas ir pranešimas: klasė turi mechanizmus, skirtus klaidoms ar netikslumams „Metamath“ duomenų bazėje ar įrodymuose aptikti ir apie juos pranešti. Tai svarbu siekiant užtikrinti duomenų bazės patikimumą ir kokybę.

Pritaikymas darbas su naršykle: Kadangi sistema planuojama naudoti HTML pagrindu veikiačioja aplinkoje buvo modifikuotas „mmverify“ paleidimo kodas ir kaip gaudomos klaidos. Šiai pakeitimai leido MM klasės funkcijas integruoti su HTML ir Brython sąsajomis, leidžiant vygditi tikrinimo procesą tiesogiai vartotojo aplinkoje be būtinybės komandinės eilutė naudojimų arba „Python“ lokalaus skripto. Taip padidinat pasiekimumą ir leidžia naudotis sistema platformų nepriklausant.

3.1.2. *Toks* klasė

„**Toks**“ klasė skirta skaityti duotą „Metamath“ tekstinį failą. Pagalbinė klasė kurios tikslas yra patikrinti ar yra prasmingas failo skaitymas. Analizuoja duoto failo duomenų bazę iš įvesties srauto ir gauna iš jo simbolių sekas, kurios vėliau bus naudojamos kituose procesuose, susijusiuose su „Metamath“ duomenų bazės analize ir verifikacija. Jos vaidmuo:

- **Srautinė analizė**: ji analizuoja tekstinį įvesties srautą – tipinį `.mm` failą, kuris susideda iš simbolių sekų, teoremų ir įrodymų.

- **Simbolių atpažinimas**: `Toks` suskaido failą į loginę simbolių seką, suprantamą programai – ji paverčia tekstinį Metamath kodą į struktūrinius vienetus, kuriuos gali naudoti `MM` klasė.

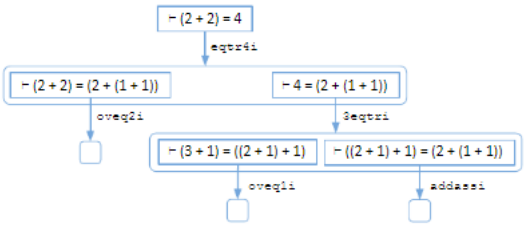
- **Sąlyginis filtravimas**: klasė sugeba ignoruoti komentarus, apdoroti makrokomandas ar tvarkyti specialius simbolius, jei jų reikalauja Metamath sintaksė.

Galima sakyti, kad jos pagrindinis darbas veikti kaip skaitytuvas, kuris parverčia pliką tekstą į duomenis, kurios galima suprasti ir padaro taip, kad su jais galima dirbti su kitais komponentais. Jei jos nebūtų nemanoma žinoti kur viena teorema prasideda, kur ji baigėsi ir kur kita prasidedą, kur yra aksiomai, hipotezės, kur prasideda įrodymo dalis.

4. Metamath vizualizavimas

„Metamath“ nepateikia savarankiškų įrankių matematinės informacijos vizualizavimui, todėl reikia naudoti kitas programavimo kalbas arba papildomus įrankius. Galėtume efektyviai vizualizuoti„Metamath“ sukurtas funkcijas ir padaryti jas suprantamas kitiems naudotojams. Šie įrankiai gali interpretuoti ir atvaizduoti matematinę struktūrą bei sąvokas, kurios yra užkoduotos„Metamath“ duomenų bazėje. Vizualizacija gali būti ypatingai naudinga sudėtingų matematinių sąryšių aiškinimui, padėti suprasti matematines sąvokas ir pateikti matematinius įrodymus intuityviai ir suprantamai.

Pavyzdžiui, 2 paveikslėlyje pateikiama, kaip galėtų atrodyti vizualizuota „Metamath“, apie kurią buvo kalbėta 1.1 skyriuje. Tai gali palengvinti matematinės informacijos suvokimą ir bendravimą tarp matematikos specialistų.



2 pav. 2 + 2 = 4 Metamath įrodymo medžio vizualizacija [RBG22]

4.1. Metamath galimi vizualizavimo būdai

- Grafiko vaizdavimas gali apimti grafikus ir diagramas, kurios atvaizduoja matematines struktūras, tokias kaip aibės, ryšiai, funkcijos arba loginės priklausomybės. Šis grafiko vaizdavimas padeda vartotojui geriau suprasti matematines Teoremas, ryšius ir modelius.

- Interaktyvus vizualizavimas leidžia vartotojui dinamiškai tyrinėti matematikos sąvokas. Tai apima interaktyvų brėžimą, objektų moduliavimą ir manipuliavimą, leidžiantį vartotojui interaktyviai eksperimentuoti su skirtingais parametrais arba įvestinėmis.

- Diagraminis įrodymas: vietoj tradicinių simbolių naudojimo įrodymams pateikti vizualizacija gali būti naudinga įrodymo žingsniams ir loginiams ryšiams pavaizduoti diagramomis. Tai padeda intuityviau suprasti loginę įrodymo struktūrą. Šiame kontekste galima išskirti kelias matematinės vizualizacijos formas:

Šiame kontekste galima išskirti kelias matematinės vizualizacijos formas:

- **Eulerio diagramos**: tai grafinės vaizdinės priemonės, kuriose aibės ir jų ryšiai vaizduojami naudojant uždaras kreives arba apskritimus. Šios diagramos gali būti naudojamos aibių operacijoms, loginiams ryšiams ir išvadoms iliustruoti aibių įrodymus.

- **Įrodymų medis**: taip pat žinomi kaip įrodymų miškai arba įrodymų grafikai, šie struktūriniai grafikai parodo teiginių ir loginių įrodinėjimo žingsnių hierarchiją ir ryšius. Mazgai žymi teiginius, o briaunos žymi loginius ryšius arba išvadų taisykles.
- **Komutacinės diagramos**: yra naudojamos tam tikrose matematikos srityse, pavyzdžiui, kategorijų teorijoje. Jos vaizduoja ryšius tarp objektų ir morfizmų, padėdamos vizualiai suprasti sąryšius ir jų struktūrą.

5. Vizualizacijos

5.1. Python

„Python“ yra aukšto lygio programavimo kalba. Šią kalbą lengva mokytis ir ja naudotis. Taip pat yra dažnai naudojama įvairiems programavimo tikslams, įskaitant interneto programavimą, duomenų analizę. Keletas žinomu savybių:

1. **Sintaksės paprastumas**: „Python“ turi aiškią ir suprantamą sintaksę, kurią lengva skaityti ir rašyti. Tai yra privalumas naujokams programuotojams.

2. **Plati standartinė biblioteka:** „Python“ turi didelę standartinę biblioteką, kuri suteikia įrankius daugeliui užduočių, tokių kaip failų tvarkymas, tinklo programavimas, teksto analizė, duomenų bazės valdymas ir kt.

3. **Didelė bendruomenė**: „Python“ turi didelę programuotojų bendruomenę, kuri aktyviai prisideda prie kalbos vystymo ir teikia įvairias bibliotekas bei modulius, kurie palengvina programavimą.

4. **Įvairios pagalbinės priemonės**: bendravimui su kitomis kalbomis, vertimui į kitas kalbas arba bendram darbui. Pavyzdžiui, galite naudoti „pyscript“ arba „Brython“ vaizdus. Abu jie skirti tekstui į „HTML“ kurti be „javascript“ pagalbos.

5.2. GUI

Grafinės vartotojo sąsajos (GUI) kūrimas gali būti daromas per internetinį (web) kontekstą.

- **Internetinės programos**: internetinių programų kūrimas dažnai susijes su trijų svarbių technologijų naudojimu: „HTML“, „CSS“ ir „Python“. „HTML“ (HyperText Markup Language) yra žymėjimo kalba, skirta internetinio turinio struktūrizavimui. Ji nustato, kaip turinys turėtų būti organizuojamas ir atvaizduojamas naršyklėje, įtraukiant tekstą, paveikslėlius, nuorodas ir kitus elementus. „CSS“ (Cascading Style Sheets) yra stilių kalba, kuri suteikia galimybę nustatyti internetinio puslapio išvaizdą, įskaitant spalvas, šriftus, pozicijas ir kitus vizualinius aspektus. Be to, „Python“ dažnai yra naudojama programų logikai ir funkcionalumui valdyti internetinėse programose.

„HTML“ yra žymėjimo kalba, naudojama organizuoti turinį internete, o „CSS“ leidžia valdyti išvaizdą ir stilius. Kuriant grafikus galima panaudoti SVG.

- **SVG**(Scalable Vector Graphics) reiškia keičiamo dydžio vektorinį grafiką. Tai žymėjimo kalba, skirta apibūdinti dvimatį grafiką XML formatu. „SVG“ vaizdus galima keisti iki skirtingų dydžių neprarandant kokybės, nes jie pagrįsti matematiniais formų aprašymais, o ne pikselių pagrįstais vaizdais. „SVG“ dažniausiai naudojama piktogramoms, iliustracijoms,

diagramoms ir kitiems grafiniams elementams žiniatinklyje. Ją palaiko visos šiuolaikinės žiniatinklio naršyklės.[23b]

- **CSS**(Cascading Style Sheets)– tai stilių lapų kalba, naudojama dokumento, žymėjimo kalba, pvz., „HTML“, pateikimui ir stiliui nustatyti.[24]

5.3. Brython

„Brython“ – tai „Python 3“ įgyvendinta kliento pusės žiniatinklio programavimo kalba, leidžianti rašyti „Python“ kodą ir paleisti jį naršyklėje. Pavadinimas „Brython“ reiškia „Browser Python“ [23a]. Ji verčia „Python“ kodą į „JavaScript“, kad ji būtų vykdoma naršyklės aplinkoje. Pagrindinės jos funkcijos:

1. **Python sintaksė**: „Brython“ leidžia rašyti kliento pusės kodą naudojant gryną „Python“ sintaksę. Tai reiškia, kad „Python“ programuotojai gali lengvai pereiti prie žiniatinklio programavimo, nes naudojama jiems gerai pažįstama kalba.

2. **Python sintaksė:** Suderinamumas su įvairiomis platformomis: kadangi „Brython“ kodas yra verčiamas į „JavaScript“, jis gali būti vykdomas beveik bet kurioje šiuolaikinėje naršyklėje, todėl programa yra labai suderinama su įvairiomis platformomis.

3. **Integracija su „HTML“ ir „CSS“**: „Brython“ sklandžiai integruojama su „HTML“ ir „CSS“, kad būtų galima kurti dinamiškas ir interaktyvias žiniatinklio programas. Taigi, kuriamas bendravimas tarp „Brython“, „HTML“ ir „CSS“ suteikia galimybę kurti dinamiškas ir interaktyvias internetines programas, naudojant „Python“ sintaksę ir išvaizdą, kurios valdomos naudojant „CSS“. Tai leidžia kurti lengvai skaitomą, tvarkingą ir efektyvų kodą, kuris gali būti vykdomas naršyklėje be jokių papildomų įrankių ar papildinių.

4. **Prieiga prie naršyklės API(Application Programming Interface)**: „Brython“ suteikia prieigą prie naršyklės API, todėl kūrėjai gali sąveikauti su dokumento objekto modeliu (DOM) ir tvarkyti įvykius „Python“ kodu. Štai keli pagrindiniai būdai, kaip galima panaudoti vartotojų kuriant:

   a. **DOM manipuliavimas**: naršyklės DOM (Document Object Model) suteikia galimybę manipuliuoti „HTML“ elementais ir jų savybėmis. Naudojant „Brython“, galima naudoti DOM funkcijas ir metodus, pavyzdžiui „querySelector“, „getElementById“, „createElement“, „appendChild“, norint kurti, keisti ar šalinti „HTML“ elementus.

   b. **Įvykių valdymas**: „Brython“ leidžia prijungti įvykius prie „HTML“ elementų naudojant „Python“ sintaksę. Galima naudoti „addEventListener“ funkciją, kad atsakytumėte į vartotojo veiksmus, tokius kaip paspaudimai, „pelė“, „žingsnis“ ir kt.

   c. **Duomenų siuntimas**: naudojant „Brython“, galima naudoti naršyklės funkcijas, siųsti duomenis į serverį ir gauti atsakymą.

d. **Kitos naršyklės funkcijos**: taip pat galima naudoti kitas naršyklės funkcijas, pavyzdžiui, darbo su slapukais („cookies“), lokalinių duomenų saugojimai („localStorage“), geografinės padėties nustatymai ir kitokios panašios galimybės.

5. **„Python“ standartinės bibliotekos palaikymas:** nors „Brython“ negali visiškai palaikyti visos „Python“ standartinės bibliotekos dėl naršyklės aplinkos apribojimų, ji įtraukia daugumą modulių, skirtų įprastoms užduotims, tokioms kaip darbas su eilutėmis, sąrašais ir žodynais.

„Brython“ suteikia alternatyvų požiūrį į tinklapių kūrimą, leisdama kurti kliento žiniatinklio programas „Python“ kalba. Tai yra didelis privalumas tiems, kurie mėgsta „Python“ ir nori naudoti savo žinias ir įgūdžius žiniatinklio programavime. Naudojant „Brython“, galima kurti interaktyvias ir dinamiškas internetines aplikacijas naudojant gerai pažįstamą „Python“ sintaksę.

5.4. SVG

„Python“ turi bibliotekų, kurios gali padėti vizualizuoti „Metamath“. Tačiau „Brython“ negali jų palaikyti. „Scalable Vector Graphics“ sutrupintai „SVG“ yra galingas ir lankstus grafinis formatas, kuris leidžia kurti interaktyvius ir dinamiškus vaizdus internetiniuose puslapiuose. Šis formatas, paremtas XML, yra idealus vizualizacijoms, nes leidžia kurti grafiką, kuris lengvai prisitaiko prie įvairių ekrano dydžių be kokybės praradimo. „SVG“ specifikacija yra atviras standartas, kurį nuo 1999 m. kuria Pasaulinio žiniatinklio konsorciumas. „SVG“. Failų vardų plėtiniai. Suteikiant informaciją per „Brython“ galima sukurti vaizdą.

Dar keli svarbus dalykai, kodėl verta naudoti „SVG“:

1. **Vektorinės grafikos esmė**: „SVG“ aprašo grafiką kaip vektorinius objektus – linijas, formos, tekstą ir t.t., naudodamas matematinę informaciją apie jų poziciją, formą ir stilių. Vektoriniai objektai yra nepriklausomi nuo raiškos, todėl jie gali būti išsaugoti ir rodomi be kokybės praradimo, net esant dideliam ar mažam ekrano dydžiui.

2. **Interaktyvumas**: „SVG“ leidžia pridėti interaktyvumo elementų į vaizdą naudojant JavaScript arba kitas technologijas. Tai reiškia, kad „SVG“ elementams galima priskirti įvykius (event listeners), kurie reaguos į vartotojo veiksmus, pavyzdžiui, paspaudimus ar pelės judesius.

3. **Animacija**: „SVG“ palaiko animacijos efektus, leisdama kurti judančius vaizdus, perėjimus ir kitus dinaminius elementus. Animacijos gali būti sukuriamos naudojant „CSS“, JavaScript arba „SVG“ animacijos metodus.

4. **Atvira standartizacija**: „SVG“ yra atvira standartizacija, kurią nuo 1999 metų kuria Pasaulinio žiniatinklio konsorciumas (W3C). Tai reiškia, kad „SVG“ yra plačiai palaikoma naršyklėse ir naudojama kaip standartinis grafinis formatas internete.

5. **Integruotas su „HTML“**: „SVG“ gali būti įterptas tiesiogiai į „HTML“ dokumentus naudojant <svg> elementą. Tai leidžia lengvai kurti ir valdyti „SVG“ turinį kartu su kitu „HTML“ turiniu.

3 pav. Pavyzdis „metamath“ vaizdavimo per „SVG“

Taigi, žinant šius elementus, „SVG“ yra puikus pasirinkimas kurti dinamiškus ir patrauklius vizualinius turinius internetiniuose puslapiuose, ypač kai reikia vizualizuoti sudėtingas duomenų struktūras ar kurti interaktyvias diagramas, grafikus ar animacijas. Naudodami „SVG“ kartu su „Brython“, galima sukurti sudėtingas ir interaktyvias vizualizacijas, kurios leis geriau suprasti ir analizuoti „Metamath“ kalbos struktūras ir sąvokas.

6. Įrankio kūrimas ir reikalavimai

Iš surinktos informacijos galima pradėti kurti įrankį vizualizuoti „Metamath“ kalbą. Naudojamos kalbos bus „Brython“(Python), „SVG“, „CSS“ ir „HTML“. Kad nereikėtų kurti įrankio, kuris patikrintų, ar tai yra logiška ir teisinga, panaudosime jau egzistuojantį. Geriausiai tinkantis „mmverify“, kuris buvo sukurtas „Python“. Jį taip pat reikia modifikuoti.

„mmverify“ buvo modifikuota „**Toks**“ klasė. Iš pradžių gavęs failą, jį perskaitydavo, tačiau mano atveju, tai netiko ir buvo pakeista taip, kad skaitytų „string“. Taip pat buvo pakeistas paleidimas, kuris vyko per terminalo eilutę. „HTML“ ir „Brython“ tokio paleidimo negali padaryti, todėl tai reikia pakeisti.

Būtent tam bus naudojama „Python“ kalba ir jos bibliotekos. Su „Python“ bibliotekomis bus lengva vizualizuoti „Metamath“. „Metamath“ forma būtų atvaizduojama hierarchiniu medžiu. Vietoje paprasto hierarchinio medžio bus naudojamas įrodymo struktūros medis – tai specifinis hierarchinio medžio tipas, atskleidžiantis, kaip vienas teiginys išvedamas iš kitų, remiantis logikos taisyklėmis. Ši struktūra leidžia suskaidyti „Metamath“ į mažesnius komponentus, kuriuos paprasta matyti ir analizuoti. Medžio viršuje esantys elementai vadinami tėviniais mazgais, o žemiau esantys – vaikais. Tokia vizualizacija leidžia vartotojui ne tik lengvai sekti įrodymų eigą, bet ir dalytis suprantama „Metamath“ logikos schema su kitais.

6.1. Įrankio kūrimo tikslai

1. **Kodo redagavimo funkcija**:
    - Mygtukas, leidžiantis redaguoti pateiktą kodą.
    - Rašymo langas „Metamath“ kodui kurti.
2. **Failų tvarkymas**:
    - Galimybė įkelti „.mm“ failą.
    - Įkelto failo turinio atvaizdavimas.
3. **Simbolių įterpimas**:
    - Simbolių mygtukai, kurie įterpia simbolius į tekstą paspaudus.
4. **Klaidų rodymas**:
    - Klaidų atvaizdavimas raudonu tekstu.

    .
5. **Grafiko figūrų spalvų valdymas**:
    - Mygtukai, keičiantys figūrų spalvas pagal reikšmes:

- „#f“ – numatyta spalva „#c0c4ff“.
- „#e“ – hipotezė numatyta spalva „#ffff80“.
- „#a“ – aksiomas – numatyta spalva „#d700d7“.
- „#a“ – teorema – numatyta spalva „#66ff8c“.
- „#com“ – komentaras – numatyta spalva „#66ff8c“.

- **Teksto spalvų valdymas grafike**. Mygtukai, keičiantys teksto spalvas pagal reikšmes.
  - „#f“ – formulė – numatyta pradinė spalva juoda.
  - „#e“ – hipotezė – numatyta pradinė spalva juoda.
  - „#a“ – aksiomas – numatyta pradinė spalva juoda.
  - „#a“ – teorema – numatyta pradinė spalva juoda.
  - „#com“ – komentaras – numatyta pradinė spalva juoda.
- **„Stroke color“ nustatymai**. Mygtukai, keičiantys kontūrų spalvas pagal reikšmes.
  - „#f“ – formulė – numatyta pradinė spalva juoda.
  - „#e“ – hipotezė – numatyta pradinė spalva juoda.
  - „#a“ – aksiomas – numatyta pradinė spalva juoda.
  - „#p“ – teorema – numatyta pradinė spalva juoda.
  - „#com“ – komentaras – numatyta pradinė spalva juoda.

6. **Diagramos valdymas**
   - SVG diagramos judinimas:
     - Klaviatūros mygtukais: „a“, „b“, „c“, „d“ ir rodyklėmis.
     - Rodykliniais mygtukais tinklalapyje.
   - Diagramos mastelio keitimas:
     - Klaviatūros mygtukais: „+“ (didinti), „–“ (mažinti).
     - Atitinkamais mygtukais tinklalapyje.
7. **Diagramos išsaugojimas:**
   - Mygtukas „Save SVG“ – diagramos išsaugojimui SVG formatu.
   - Mygtukas „Save PNG“ – diagramos išsaugojimui PNG formatu.
8. **Pagalba**:
   - Mygtukas, atveriantis pagalbos tekstą.

6.2. Sukurti failai

Projekto metu buvo sukurti keli atskiri „Python“ kalbos moduliai. Kiekvienas jų atsakingas už tam tikrą sistemos dalį – nuo duomenų apdorojimo iki grafinės vizualizacijos ir vartotojo sąveikos.

- „**svg_viewbox_controller.py**“ –šis modulis valdo *SVG* grafikos vaizdą. Jis leidžia keisti mastelį, perkelti vaizdą bei automatiškai pritaikyti *SVG* objektą prie naršyklės lango naudojant „*viewBox*“ savybes.
- „**mmverify_on_html.py**“ – tai naršyklei pritaikyta „mmverify“ versija. Ji tikrina „Metamath“ įrodymus ir identifikuoja klaidas įrodymo žingsniuose.
- „**function.py**“ – šis modulis apdoroja įrodymus iš MM failų. Naudojant „mmverify_on_html“, tikrinamas turinys, kuris vėliau suskaidomas į komponentus, reikalingus vizualizacijai.
- „**draw.py**“ – pagrindinis grafikos generavimo modulis. Jis kuria vizualines diagramas pagal apdorotus duomenis: nustato spalvas, formas, tekstus ir bendrą elementų išdėstymą.
- „**save_and_load.py**“ – leidžia vartotojui išsaugoti sukurtas diagramas arba atsisiųsti originalius MM failus.
- „**download_svg.py**“ – skirtas sugeneruoto SVG vaizdo atsisiuntimui, kad vartotojas galėtų išsaugoti vaizdą savo įrenginyje.
- „**write.py**“ – apdoroja vartotojo įvestį tekstinėje redagavimo aplinkoje ir palaiko ryšį su kitais moduliais.
- „**main.py**“ – pagrindinis programos failas, kuris sujungia visus modulius į vientisą sistemą. Jis koordinuoja duomenų perdavimą tarp modulių, ypač tarp „function“ ir „draw“, kad visa sistema veiktų sklandžiai.

6.3. Darbo eiga

Norint sukurti tikslią ir efektyvią įrodymų vizualizacijos sistemą, pirmiausia reikėjo išsamiai susipažinti su „*Metamath*“ kalba, jos struktūra bei esamais pagalbiniais įrankiais. Atlikus įrankių analizę, buvo pasirinktas „mmverify.py“ – patikros įrankis, parašytas „Python“ kalba, leidžiantis automatiškai tikrinti įrodymų teisingumą.

Šis įrankis buvo modifikuotas: atnaujintas paleidimo mechanizmas ir klaidų grąžinimo sistema, kad jį būtų galima naudoti internetinės naršyklės aplinkoje. Tam pasitelkta „Brython“ technologija, kuri leidžia vykdyti „Python“ kodą tiesiogiai naršyklėje ir užtikrina sąveiką su HTML elementais. Tai sudarė techninį pagrindą interaktyviai vartotojo sąsajai kurti.

Buvo sukurta redagavimo aplinka, leidžianti vartotojui įvesti arba koreguoti „Metamath“ kalba parašytą kodą tiesiogiai naršyklėje.

Įrodymų vizualizacija paremta medžio struktūros principu – tai leidžia aiškiai ir nuosekliai atvaizduoti kiekvieną įrodymo žingsnį. Siekiant padidinti aiškumą, komentavimo galimybė buvo integruota tiesiai į vizualinę įrodymo eigą, leidžiant vartotojui lengvai sekti ne tik žingsnius, bet ir jų paaiškinimus.

6.3.1. Vartotojo sąsajos langas

Projekte buvo sukurtas paprastas ir patogus vartotojo sąsajos langas, skirtas „*Metamath*“ kodo rašymui. Šiame lange vartotojas gali:

- Rašyti ir redaguoti kodą,
- Matyti eilučių numeraciją,
- Stebėti žymeklį bei savo buvimo vietą faile.

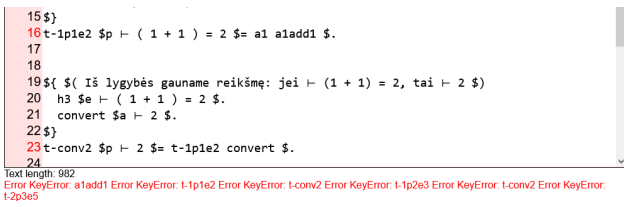
Langas taip pat turi įdiegtą simbolių skaičiuoklę ir klaidų aptikimo sistemą. Įvedus neteisingą įrodymo fragmentą, po redagavimo sritimi automatiškai pateikiamas klaidos pranešimas, o klaidingų eilučių numeriai pažymimi raudonai, kad vartotojui būtų lengviau identifikuoti ir ištaisyti klaidas.

Jei vartotojas nori įterpti komentarus į įrodymą, jie turi būti įrašyti tiesiogiai į įrodymo seką po simbolių žymos „$=“, laikantis „Metamath“ sintaksės. Tokiu būdu komentarai natūraliai integruojami į vizualizaciją, o tekstinė ir grafinė pateikimo dalys išlieka nuoseklios.

Be to, siekiant palengvinti matematinio kodo rašymą, virš redagavimo lauko buvo įdiegta simbolių juosta – joje pateikti visi dažniausiai naudojami simboliai, kuriuos sunku rasti standartinėje klaviatūroje. kuria yra : „

`+, =, ↔, ⊢, →, ¬, ⊥, ∧, ∨, ∀, ∃, ℝ, ⊆, ⊂, ∅, ∪, ∩, ∈, ∉`

“ Tai leidžia greitai įterpti reikiamus simbolius vienu paspaudimu, nepertraukiant darbo eigos.



4 pav. Klaida tarp „a1“ ir „add1“ nėra tarpo

6.3.2. Vizualizacijos

Kaip jau buvo minėta anksčiau, vizualizacijos funkcionalumas yra įgyvendintas atskirame faile „draw.py“. Šiame faile apibrėžtos visos pagrindinės grafinio atvaizdavimo valdymo funkcijos, įskaitant naudotojo sąveiką su sąsajos elementais, tokiais kaip mygtukai. Viena iš funkcijų leidžia

keisti objektų spalvas – galima grąžinti jas į numatytąsias (angl. default) arba pritaikyti vartotojo pasirinktą spalvų schemą. Buvo priimtas sprendimas, kad net ir tuo atveju, kai įvestyje yra klaidų, sistema vis tiek bandys sugeneruoti vizualizaciją. Jei konkretus elementas – pavyzdžiui, funkcija ar įrodymo žingsnis – yra neteisingas arba neatpažintas, jo vietoje vizualizacijoje pateikiamas tuščias blokas. Tokiu būdu išlaikomas bendras vaizdo vientisumas, o vartotojui tampa lengviau pastebėti, kur įvyko klaida. Ypač jei klaidos yra tarp simbolių „= “ir„.“. Siekdami aiškumo, kiekvieną įrašo komponentą tekstinėje išraiškoje žymime specialiais simboliais:

- „$a“ – aksioma.
- „$e“ – hipotezė.
- „$p“ – teorema arba įrodymo žingsnis.
- „$f“ – formulė ar kitas formalus vieneta.

Iš viso galimos keturios skirtingos vizualizacijos formos, leidžiančios lanksčiai prisitaikyti prie vartotojo poreikių. Nors kai kurios iš jų gali atrodyti panašiai, jos suteikia skirtingus įžvalgų lygius, priklausomai nuo to, ar vartotojui reikia išsamios, ar supaprastintos informacijos peržiūros.

Vartotojui taip pat suteikiama galimybė pasirinkti, kaip bus formuojami stačiakampiai, reprezentuojantys įrodymo žingsnius: jie gali būti vienodo fiksuoto dydžio, arba automatiškai prisitaikantys prie teksto ilgio. Be to, galima individualiai nustatyti kiekvienos figūros:

- užpildo spalvą.
- kontūro spalvą.
- teksto spalvą viduje.

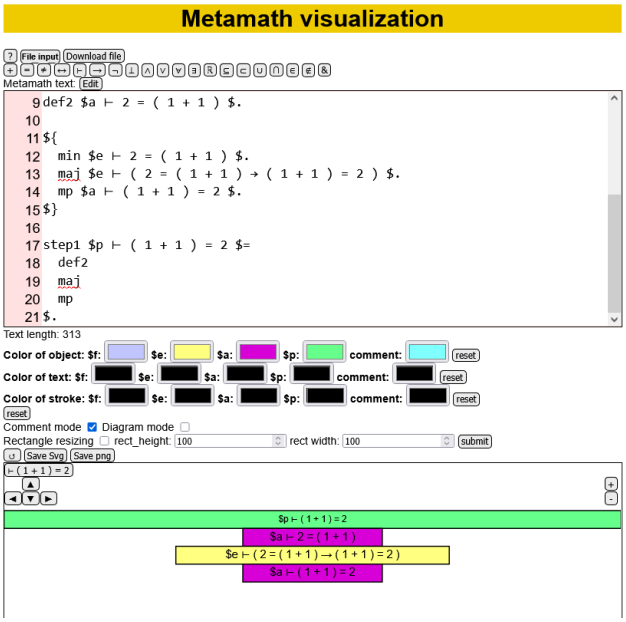
Visa ši funkcionalumo įvairovė leidžia vartotojui lanksčiai ir interaktyviai valdyti vizualinį įrodymų pateikimą, atsižvelgiant į savo poreikius ir estetinius pasirinkimus. Taip pat buvo įdiegta galimybė išsisaugoti sukurtas diagramas atskirai – kiekvieną įrodymą galima eksportuoti kaip atskirą failą. Kadangi kai kurios diagramos gali būti didesnės nei vizualizacijos langas, buvo sukurta interaktyvi naršymo sistema, leidžianti lengvai judėti po diagramos vaizdą: Naudojant klaviatūros rodyklių klavišus, galima judėti į visas keturias kryptis. Spaudžiant mygtukus tinklapyje, galima perkelti vaizdą tarsi „stumdant“ jį. Naudojant klavišus „+“ ir „-“, galima keisti mastelį – padidinti arba sumažinti diagramos vaizdą.

Šios funkcijos užtikrina, kad net ir sudėtingi ar didelės apimties įrodymai būtų lengvai peržiūrimi, patogiai valdoma vizualizacija ir intuityvi naudotojo sąsaja.

6.4. Kas jau sukurta

Darbo metu buvo realizuoti šie pagrindiniai komponentai ir funkcionalumai:

- **Metamath failų analizės ir interpretavimo mechanizmas**: Sukurtas parseris, galintis perskaityti .mm formato failus, išskirti simbolius, aksiomas, teoremas bei jų tarpusavio ryšius. Ši sistema užtikrina sintaksinį tikslumą ir parengia duomenis vizualizacijai.
- **„mmverify“ integracija**: Integruota patobulinta „mmverify.py“ versija, kuri leidžia automatizuotai tikrinti teiginių logiškumą bei taisyklingą teoremų pagrindimą. Sistema pateikia klaidų pranešimus, kai įrodymai neatitinka formalių taisyklių.
- **Brython aplinkos pritaikymas**: „Brython“ biblioteka pritaikyta taip, kad naudotojo sąsaja būtų vykdoma tiesiogiai naršyklėje be papildomų įdiegimų. Įgyvendintas kodų vertimas realiu laiku ir bendradarbiavimas su HTML elementais.
- **Vizualizacijos sistema su SVG**: Sukurtas vizualizavimo modulis, kuris vaizduoja įrodymų žingsnius grafiškai, naudojant SVG formatą. Įrodymai pateikiami medžio struktūra, leidžiančia lengvai sekti teiginių priklausomybes ir jų eiliškumą.
- **Interaktyvios vartotojo sąsajos komponentai**: Implementuoti naudotojo sąsajos valdikliai, leidžiantys didinti, mažinti, perkelti vizualizacijos langą, aktyvuoti komentarus, bei keisti atvaizdavimo režimus (pvz., su arba be įrodymo žingsnių).
- **Komentarų vizualinis pateikimas**: Pridėta galimybė naudotojui matyti komentarus, aprašytus Metamath faile, kartu su atitinkamais įrodymo žingsniais. Tai sustiprina supratimą ir leidžia naudotojui lengviau orientuotis informacijoje.
- **Eksportavimo funkcijos**: Įdiegta galimybė atsisiųsti sugeneruotą vizualizaciją SVG arba PNG formatais, leidžianti vartotojui dalintis rezultatais ar naudoti juos kitoje aplinkoje (pvz., pristatymuose ar dokumentuose).



5 pav. Enter Caption

Rezultatai ir išvados

Rezultatai

Šio baigiamojo darbo metu buvo atlikti šie veiksmai:

1. Išanalizuota Metamath kalba ir jos veikimo principai, įskaitant pagrindinius kalbos komponentus, tokius kaip aksiomos, hipotezės, teoremos bei jų tarpusavio ryšiai.
2. Peržiūrėti ir įvertinti egzistuojantys Metamath kalbos įrankiai, tarp jų – „mmverify.py“ ir „Yamma“ redagavimo aplinka.
3. Išanalizuota „Brython“ kalba, kuri leido įterpti Python kodą į HTML struktūrą naršyklėje, taip sukuriant interaktyvią naudotojo sąsają be serverinės pusės.
4. Naudota „SVG“ (Scalable Vector Graphics) technologija vizualinių elementų kūrimui ir interaktyvių diagramų pateikimui.
5. Sukurtas prototipas – įrankis, leidžiantis naudotojui įkelti Metamath .mm failus, vizualizuoti juose pateiktus įrodymus ir keisti diagramos elementus (objektų spalvas, kontūrus, tekstus).
6. Įgyvendintas interaktyvumas – pridėtos klaviatūros ir mygtukų sąsajos, leidžiančios naršyti diagramą, ją didinti, mažinti ar perkelti.
7. Įdiegta funkcija komentarų atvaizdavimui ir galimybė perjungti vizualizavimo režimus: detalus su žingsniais, su ar be komentarų, santraukinis vaizdavimas.
8. Įgyvendinta galimybė atsisiųsti vizualizaciją SVG ir PNG formatais.
9. Dokumentuotas sukurtas funkcionalumas, pateikti tolesni kūrimo planai – galimybė išplėsti sprendimą teoremų analizei ir integruoti su kitais matematinės logikos įrankiais.

Išvados

Apibendrinant, šis darbas sėkmingai įgyvendino pagrindinį tikslą – sukurtas intuityvus vizualizavimo įrankis, skirtas „ *Metamath*“ kalbos struktūrai ir įrodymų logikai suprasti bei analizuoti. Naudojant šiuolaikines žiniatinklio technologijas, tokias kaip „Brython“ ir SVG, reikšmingai pagerinta naudotojo patirtis, palyginti su tradiciniais Metamath įrankiais. Vizualiai interaktyvus informacijos pateikimas leidžia vartotojams lengviau suvokti sudėtingus loginius ryšius ir lanksčiai valdyti informacijos srautą pagal savo poreikius.

Be to, įrankio naudojimas be papildomų diegimų, naudojimas per naršykle suteikia didesnes galimybes. Ateityje vizualizavimo logiką būtų galima tobulinti, plečiant informaciją apie kiekviename žingsnyje panaudotus blokus bei teiginius. Tai suteiktų dar gilesnį supratimą apie įrodymų struktūrą ir loginius ryšius .

Nors sukurtas įrankis jau dabar yra vertingas pagalbininkas, tolimesnės plėtros galimybės – tokios kaip semantinis žymėjimas ar integracija su kitais Metamath šaltiniais – gali dar labiau išplėsti jo funkcionalumą ir svarbą bendruomenei. Galima teigti, kad šis įrankis turi potencialo tapti reikšminga priemone tiek pradedantiesiems, siekiantiems perprasti Metamath pagrindus, tiek pažengusiems vartotojams, atliekantiems sudėtingus įrodymų tyrimus ir verifikaciją.

Literatūra

[23a] Brython. https://brython.info, 2023.

[23b] SVG Animations Level 2. https://svgwg.org/specs/animations/#intro, 2023.

[24] CSS: Cascading Style Sheets. https://developer.mozilla.org/en-US/docs/Web/CSS, 2024.

[Car12] Mario Carneiro. yamma. https://github.com/glacode/yamma/tree/master, 2012. tikrinta 2023-08-12.

[gla12] glacode. mmj2. https://github.com/digama0/mmj2, 2012. tikrinta 2023-06-12.

[Ies22] Igor Ieskov. metamath-lamp. https://github.com/expln/metamath-lamp, 2022. tikrinta 2023-07-12.

[Lev] Raph Levien. mmverify. https://github.com/david-a-wheeler/mmverify.py. tikrinta 2023-06-12.

[Meg05] Norman Megill. Metamath Proof Explorer. https://us.metamath.org/mpeuni/mmset.html, 2005. tikrinta 2021-06-04.

[Meg21] Normanas Megila. Higher-Order Logic Explorer. https://us.metamath.org/holuni/axrep.html, 2021. tikrinta 2021-08-04.

[Nor23a] Rimas Norvaiša. Matematika. https://www.vle.lt/straipsnis/matematika/, 2023. tikrinta 2023-07-12.

[Nor23b] Rimas Norvaiša. Pitagoro teorema. https://www.vle.lt/straipsnis/pitagoro-teorema/, 2023.

[RBG22] Reuel R. Ribeiro, Valmir C. Barbosa, and Flavio B. Gonzaga. *Graph based analysis of mathematical knowledge structure on Metamath*. PhD thesis, Núcleo de Ciência da Computação, Universidade Federal de Alfenas, 2022.

Priedas

Priedai

Programos išeities kodas patalpintas viešoje „GitHub” repozitorijoje. Norėdami jį pasiekti, naudokite šią nuorodą: `https://github.com/RasUrb/Metamath-vizualizacija.git`.

Programos atsisiuntimas

Atidarykite nuorodą į „GitHub” repozitoriją.
Paspauskite mygtuką **„Code”** ir pasirinkite **„Download ZIP”**.
Atsisiuntę ZIP archyvą, išskleiskite jį į pasirinktą vietą kompiuteryje.

Programos paleidimas

Norėdami paleisti programą, atlikite vieną iš šių veiksmų:

- Dukart spustelėkite failą `serveris.py` (jeigu jūsų sistema tai leidžia ir yra tinkamai sukonfigūruota).
- Arba paleiskite programą per terminalą, įvesdami šią komandą kataloge, kur yra išskleistas projektas:

  ```
  python starter.py
  ```

Prieš paleidžiant įsitikinkite, kad jūsų sistemoje įdiegtas `Python` interpretuotojas (rekomenduojama versija: 3.8 ar naujesnė).