КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп’ютерні науки

на тему:

СИСТЕМА ДЛЯ РОЗВ’ЯЗАННЯ ГЕОМЕТРИЧНИХ ЗАДАЧ НА ОСНОВІ СЕКВЕНЦІЙНОГО РЕНОМІНАТИВНОГО ЧИСЛЕННЯ

Виконав студент 4-го курсу бакалаврату

Михайло ШВЕЦЬ

(підпис)

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Віталій ЗУБЕНКО

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

«_**05**_» ___**_червня**____ 20**23** р.,

протокол № _**18**_

Завідувач кафедри

Микола НІКІТЧЕНКО

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 62 сторінки, 9 ілюстрацій, 13 джерел посилань.

АВТОМАТИЗАЦІЯ ДОВЕДЕННЯ ТЕОРЕМ, ГЕОМЕТРІЯ, ПРОЄКТУВАННЯ, РЕНОМІНАТИВНА ЛОГІКА, РОЗРОБКА, СЕКВЕНЦІЙНЕ ЧИСЛЕННЯ, СИСТЕМА РІВНЯНЬ ТА НЕРІВНОСТЕЙ.

Об’єктом роботи є процес доведення геометричних фактів за допомогою програмного продукту для автоматичного доведення теорем на основі секвенційного реномінативного числення.

Метою роботи є розробка програмного засобу для доведення геометричних теорем на основі секвенційного реномінативного числення.

Методи розробки: система рівнянь, реномінативне числення, принципи SOLID. Інструменти розробки: мова програмування Java, середовище програмування IntelliJ IDEA.

Результати роботи: проведено аналіз наявних продуктів з даної тематики на ринку і зроблено висновок, що продуктів, які поглиблено спеціалізуються на геометрії, наразі немає, проведено аналіз найчастіше вживаних інструментів розробки та обрано оптимальні варіанти, які і були використані при написанні програми, розроблено детальну архітектуру програмної системи, а також формально записано алгоритми потрібні для її роботи, внаслідок чого швидкість написання коду збільшилась, здійснено програмну реалізацію продукту з урахуванням результатів отриманих раніше.

Програмний продукт «Система для розв’язання геометричних задач на основі секвенційного реномінативного числення» може застосовуватися для розробки інших програмних продуктів, під час виконання різних інженерних розрахунків або в навчальному процесі в шкільному курсі геометрії.

ЗМІСТ

ВСТУП

На сьогоднішній день математична логіка залишається важливою та актуальною дисципліною в сучасному світі. Вона використовується в багатьох галузях, включаючи штучний інтелект, філософію, мовознавство, криптографію, теорію баз даних, математику та інші.

Одним з основних напрямків застосування математичної логіки є комп'ютерні науки. Логічні системи, такі як пропозиційна логіка і предикатна логіка, використовуються для формалізації та аналізу алгоритмів, розробки програмного забезпечення, верифікації програм і апаратних засобів, а також у сфері штучного інтелекту для розуміння та моделювання розумових процесів.

Геометрія в сучасному світі і досі застосовується вкрай широко у багатьох сферах, в тому числі в різних напрямках комп’ютерних технологій. Такі проблеми як комп’ютерний зір вимагають ґрунтовних знань алгоритмів, які засновані на геометричних принципах. Тому було прийняте рішення зосередитись саме на цій тематиці, оскільки, в разі продовження розробки продукту, можливості для практичного застосування отриманих результатів не вичерпаються.

Метою написання цієї роботи є аналіз можливості реалізації алгоритмів пошуку доведень на прикладі конкретної тематики – геометричних задач. Під час її виконання було спроєктовано та розроблено систему, яка, використовуючи винайдене ще до появи обчислювальних систем секвенційне числення, здатна будувати висновки на основі відомих фактів. Також було запропоновано декілька можливостей для подальшого розширення продукту.

Одним із прикладів практичного застосування є галузь шкільної освіти. Вкрай важливо розвинути у підростаючого покоління вміння не тільки розв’язувати задачі, а й робити правильні висновки в будь-яких інших ситуаціях, що потребують аналізу та процедури логічного виведення. В перспективі результати будуть корисні при розробці автоматизованих систем для синтезу нових знань і, можливо, зможуть вплинути на створення першого у світі сильного

штучного інтелекту, який зможе розв’язати будь-яку задачу, яку здатна розв’язати людина.

Для розробки програмного забезпечення було використано мову Java та середовище розробки IntelliJ IDEA.

РОЗДІЛ 1 НАЯВНІ РІШЕННЯ

Перед виконанням даної роботи було проведено аналіз проблеми доведення геометричного факту виходячи з набору відомих фактів, теорем та аксіом. Як виявилося в цій галузі наразі конкурентами є два продукти від різних компаній, проте основна увага зосереджена на математичних проблемах, а не на геометричних. Якщо в майбутньому буде продовжуватися розробка та тему цієї роботи, то досвід цих компаній можна буде успішно застосувати для покращення та оптимізації продукту.

1.1 Математичний обчислювач Symbolab

Із тексту на веб-сайті продукту Symbolab – це розширений інструмент математичної освіти. Він дозволяє користувачам вивчати, практикувати та відкривати різні математичні теми використовуючи математичні позначення та наукову нотацію, а також текст. Symbolab надає автоматичні, покрокові розв’язки до проблем алгебри, тригонометрії та аналізу, покриваючи теми від середньої до старшої школи. Symbolab пропонує широкий асортимент розумних калькуляторів, які включають в себе: рівняння, системи рівнянь, нерівності, інтеграли, похідні, границі, дотичні, тригонометричні рівняння, функції тощо. Заявлена ціль цього сайту – зробити науковий контент загальнодоступним за допомогою розширення простору даних на наукові нотації, вирази, рівняння та формули. Це досягається за допомогою застосування алгоритмів машинного навчання щоб зрозуміти значення та суть запитів [8].

1.2 Система WolframAlpha

Wolfram|Alpha — база знань і набір обчислювальних алгоритмів (англ. computational knowledge engine). Обчислює відповідь, ґрунтуючись на власній базі знань, яка містить дані з математики, фізики, астрономії, хімії, біології, медицини,

історії, географії, політики, музики, кінематографії, а також інформацію про відомих людей та інтернет-сайти.

Довгострокова ціль Wolfram|Alpha – зробити всі систематичні знання миттєво обчислювальними та доступними кожному.

Його місія – зібрати та проаналізувати всі об’єктивні дані, реалізувати всі відомі моделі та алгоритми та зробити можливим обчислити все, що може бути обчисленим стосовно чого завгодно. Робота ґрунтується на досягненнях науки та інших систематизацій знань, щоб забезпечити одне джерело, на яке можна покластися всім для остаточних відповідей на фактичні запитання.

Wolfram|Alpha покликаний принести знання та можливості експертного рівня найширшому колу людей, включаючи всі професії та рівні освіти.

Команда працює над тим, щоб приймати абсолютно вільну форму введення та функціонувати як машинний інтелект, що генерує потужні результати й презентує їх з максимальною чіткістю.

Енергійно розвиваючись протягом багатьох десятиліть, Wolfram|Alpha є амбітним та довгостроковим інтелектуальним зусиллям, яке розробники сподіваються розвивати з роками, надаючи все більші можливості.

З командою світового класу та участю провідних експертів у безлічі галузей ведеться постійна праця, щоб створити досягнення XXI століття, яке стане великим інтелектуальним успіхом [9].

РОЗДІЛ 2 ВИКОРИСТАНІ ЗАСОБИ

2.1 Мова програмування Java

Java є мовою програмування, яку використовують розробники програмного забезпечення для створення різноманітних додатків для комп'ютерів, смартфонів, планшетів та інших розумних пристроїв.

Java є платформонезалежною мовою програмування, чим вона відрізняється від інших мов, таких як C, C++ тощо, які компілюються під конкретні платформи. Платформа - це апаратне або програмне середовище, на якому виконується програма. Існують два типи платформ: програмно-засновані та апаратно-засновані. Java надає програмно-засновану платформу. Платформа Java відрізняється від більшості інших платформ тим, що вона є програмно-заснованою платформою, яка працює на базі інших апаратно-заснованих платформ. Вона складається з двох компонентів:

– Середовище виконання (Runtime Environment)

– API (Application Programming Interface)

Java-код може бути виконаний на різних платформах, наприклад, Windows, Linux, Sun Solaris, Mac/OS і т. д. Java-код компілюється компілятором і перетворюється на байт-код. Цей байт-код є платформонезалежним, оскільки його можна виконувати на різних платформах, тобто «один раз написав – запускай де завгодно».

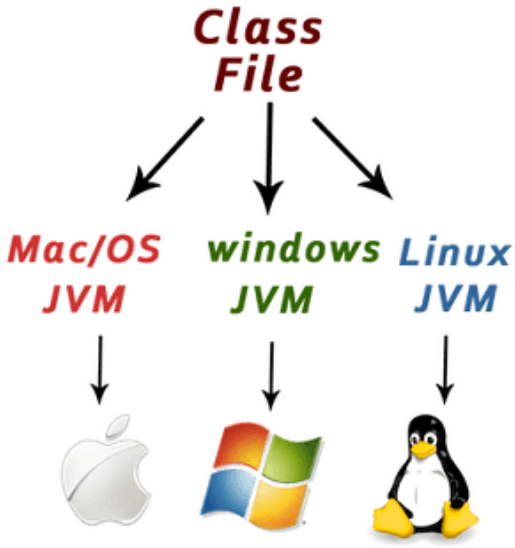


Рисунок 1.1 – мультиплатформеність мови Java

Для виконання програм на Java достатньо встановити Java Runtime Environment (JRE), завантаживши його з веб-сайту http://www.oracle.com. JRE доступний для різних операційних систем, таких як Linux (x86, x64), Mac OS (x64), Solaris і Windows (x86, x64), що дозволяє працювати з кодом Java на широкому спектрі комп'ютерів та операційних систем.

JRE забезпечує безпечну та зручну роботу програм на Java, захищаючи користувачів від несанкціонованого втручання в ресурси їх персональних комп'ютерів з боку стороннього коду Java. Достатньо періодично оновлювати JRE, остання версія - JRE 8 Update 171. Вбудовані технології безпеки Java включають широкий набір механізмів API (Application Programming Interface) та додаткові інструменти, які використовують відомі та надійні алгоритми та протоколи безпеки.

Це включає в себе використання криптографічних механізмів захисту інформації, інфраструктуру відкритих ключів, безпечну комунікацію, аутентифікацію та контроль доступу.

Основними перевагами цієї мови над іншими є:

– Легкість розуміння синтакису. Є однією з найкращих мов програмування для написання та розуміння написаного коду.

– Є об’єктно-орієнтованою мовою. Це дозволяє використовувати принципи SOLID при розробці програм, що збільшує швидкість написання та відкриває більше можливостей для аналізу та рефакторингу.

– Відсутність низькорівневих конструкцій, наприклад вказівників. Це знижує вірогідність настання витіків пам’яті а також випадкового пошкодження даних внаслідок неавторизованого доступу. Низькорівневі функції, наприклад збір сміття, виконуються JVM автоматично.

– Мова програмування Java є мультиплатформеною. Через наявність вбудованої віртуальної машини, програми написані на Java не зобов'язані враховувати особливості системи на якій вони запущені.

– У Java реалізована підтримка мультипотоковості та паралельного виконання. Це дозволяє реалізовувати системи розподілених обчислень [10].

2.2 Середовище розробки IntelliJ IDEA

Для розробки було обрано середовище IntelliJ IDEA, оскільки воно має ряд переваг над своїми аналогами (рисунок 1.2).

Рисунок 1.2 – особливості середовища IntelliJ IDEA

Середовище відоме як найперше з ґрунтовною підтримкою швидкої реорганізації програмного коду. IntelliJ IDEA пропонує широкий набір автоматизованих рефакторингів коду, які значно підвищують продуктивність. Наприклад, при перейменуванні класу IDE оновить всі посилання на цей клас у проєкті. IntelliJ IDEA може зрозуміти, яку частину планується рефакторити, і запитує підтвердження, якщо можливі декілька варіантів.

IntelliJ IDEA Ultimate є розширеною версією більшості IDE, які базуються на платформі IntelliJ. Якщо включені додаткові плагіни для мов програмування, IntelliJ IDEA Ultimate включає підтримку всіх технологій, що доступні у більш специфічних IDE, таких як PyCharm, WebStorm, PHPStorm та інші.

IntelliJ IDEA надає середовище, орієнтоване на розробника, дозволяючи сконцентруватися на функціональних задачах у той час, як IntelliJ бере на себе виконання рутинних операцій. Воно слідкує за контекстом і автоматично викликає необхідні інструменти, щоб допомогти мінімізувати ризик перебоїв у процесі розробки.

IntelliJ IDEA надає набір інспекцій, які є вбудованими засобами статичного аналізу коду. Вони допомагають знайти потенційні помилки, виявити непотрібний код, виявити проблеми з продуктивністю та покращити загальну структуру коду.

IntelliJ IDEA надає вбудований JVM-дебагер. Він дозволяє отримувати та аналізувати інформацію під час виконання програми, що є корисним для діагностики проблем та отримання глибшого розуміння роботи програми. Він дозволяє:

– Призупинити виконання програми для аналізу її поведінки за допомогою точок зупинки. Різні типи точок зупинки, разом з умовами та фільтрами, дозволяють вказати саме той момент, коли програма потребує призупинки.

– Змінювати значення змінних, обчислювати вирази та інше, що дозволяє маніпулювати станом програми.

– Аналізувати значення змінних, стеки викликів, стани потоків та інше.

– Контролювати поетапне виконання програми.

IntelliJ IDEA має повнофункціональну інтеграцію з Gradle та Maven, що дозволяє автоматизувати процес збирання, упаковки, виконання тестів, розгортання та інші дії. При відкритті існуючого проєкту Gradle або Maven або створенні нового, IntelliJ IDEA автоматично виявляє та завантажує всі необхідні репозиторії та плагіни, тому практично не потрібно нічого налаштовувати і можна зосередитися виключно на процесі розробки. Можливо змінювати файли build.gradle та pom.xml безпосередньо з редактора та налаштувати IDE так, щоб автоматично синхронізувати всі зміни з конфігураціями збирання.

Програмний продукт розповсюджується з двома редакціями: Community Edition та Ultimate Edition. Перша редакція є безкоштовною з урізаним

функціоналом, проте Ultimate Edition є доступною для безкоштовного користування особам зі студентською підпискою [11].

РОЗДІЛ 3 ТЕОРЕТИЧНІ ВІДОМОСТІ

3.1 Секвенційне реномінативне числення

В теорії доведень та математичній логіці секвенційне числення є сімейством формальних систем, які об'єднує стиль виведення і деякі формальні властивості. Перша система секвенційного числення була розроблена Г. Генценом. в 1934/1935 р. в якості інструмента дослідження природньої дедукції в логіках першого порядку [2][3].

Нехай $\Gamma \vDash \Delta$ довільний тавтологічний наслідок і $\Gamma = \{\Phi_1, \dots, \Phi_n\}$, $\Delta = \{\Psi_1, \dots, \Psi_n\}$. Секвенцією називається слово $[\Gamma \Rightarrow \Delta\,]$ ($[\,\Rightarrow \Delta]$), яке представляє імплікацію $\Phi_1 \wedge \dots \wedge \Phi_n \to \Psi_1 \vee \dots \vee \Psi_n$(формулу $\Psi_1 \vee \dots \vee \Psi_n$). Квадратні дужки – це метасимволи, які підкреслюють, що секвенція формально не є імплікацією чи формулою, вона тільки синтаксично представляє їх. Секвенція $[\Gamma \Rightarrow \Delta\,]$ виконується, якщо її імплікація є тавтологією, тобто має місце тавтологічний наслідок $\Gamma \vDash \Delta$. Секвенції, в яких ліва і права частини мають не порожній перетин, називаються замкненими. Вони завжди виконуються.

Г. Генцен запропонував спеціальне повне секвенційне числення (СЧ) для секвенцій, що виконуються (а значить і для логічних наслідків !). Тому його іноді називають – генценівським. Аксіомами числення виступають замкнені секвенції вигляду $[\{\Phi\} \cup \Gamma \Rightarrow \Delta \cup \{\Phi\}\,]$, формули яких не містять зв'язок. Правилами виведення (ПВ) в СЧ є наступні чотири правила:

ПВ1) $[\Gamma \Rightarrow \Delta \cup \{\Phi\}] \vdash [\,\{\neg\Phi\} \cup \Gamma \Rightarrow \Delta]$

ПВ2) $[\{\Phi\} \cup \Gamma \Rightarrow \Delta] \vdash [\,\Gamma \Rightarrow \Delta \cup \{\neg\Phi\}]$

ПВ3) $[\{\Phi\} \cup \Gamma \Rightarrow \Delta], [\{\Psi\} \cup \Gamma \Rightarrow \Delta] \vdash [\,\{\Phi \vee \Psi\} \cup \Gamma \Rightarrow \Delta]$

ПВ4) $[\Gamma \Rightarrow \Delta \cup \{\Phi, \Psi\}] \vdash [\,\Gamma \Rightarrow \Delta \cup \{\Phi \vee \Psi\}]$

Секвенція – вивідна в СЧ, якщо вона має доведення з аксіом. Для зменшення висоти секвенційних дерев і скорочення відповідних доведень, СЧ можна доповнити додатковими правилами виведення, які відповідають решті зв'язок $\wedge, \to, \leftrightarrow$.

Доведення теорем в СЧ будуються методом «зверху-вниз», тобто з кінця. Спочатку кінцеву секвенцію намагаємося подати як результат застосування підходящого правила виведення і певних секвенцій-аргументів. Потім аналогічно діємо стосовно вибраних секвенцій-аргументів і так до тих пір, поки не дістанемося замкнених секвенцій або секвенцій без зв'язок [1].

Правилами виведення реномінативних неокласичних логік є правила успадковані з пропозиційного рівня, а також такі:

$$\left[\left\{R_{\bar{x}}^{\bar{v}}(A)\right\} \cup \Gamma \Rightarrow \Delta\right] \vdash \left[\left\{R_{z,\bar{x}}^{z,\bar{v}}(A)\right\} \cup \Gamma \Rightarrow \Delta\right]$$

$$\left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}}(A)\right\}\right] \vdash \left[\Gamma \Rightarrow \Delta \cup \left\{R_{z,\bar{x}}^{z,\bar{v}}(A)\right\}\right]$$

$$\left[\left\{R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}} (A)\right\} \cup \Gamma \Rightarrow \Delta\right] \vdash \left[\left\{R_{\bar{x}}^{\bar{v}}\left(R_{\bar{y}}^{\bar{w}}(A)\right)\right\} \cup \Gamma \Rightarrow \Delta\right]$$

$$\left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}} (A)\right\}\right] \vdash \left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}}\left(R_{\bar{y}}^{\bar{w}}(A)\right)\right\}\right]$$

$$\left[\left\{\neg R_{\bar{x}}^{\bar{v}}(A)\right\} \cup \Gamma \Rightarrow \Delta\right] \vdash \left[\left\{R_{\bar{x}}^{\bar{v}}(\neg A)\right\} \cup \Gamma \Rightarrow \Delta\right]$$

$$\left[\Gamma \Rightarrow \Delta \cup \left\{\neg R_{\bar{x}}^{\bar{v}}(A)\right\}\right] \vdash \left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}}(\neg A)\right\}\right]$$

$$\left[\left\{R_{\bar{x}}^{\bar{v}}(A)\right\} \cup \Gamma \Rightarrow \Delta\right], \left[\left\{R_{\bar{x}}^{\bar{v}}(B)\right\} \cup \Gamma \Rightarrow \Delta\right] \vdash \left[\left\{R_{\bar{x}}^{\bar{v}}(A \vee B)\right\} \cup \Gamma \Rightarrow \Delta\right]$$

$$\left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}}(A), R_{\bar{x}}^{\bar{v}}(B)\right\}\right] \vdash \left[\Gamma \Rightarrow \Delta \cup \left\{R_{\bar{x}}^{\bar{v}}(A \vee B)\right\}\right]$$

Процедура побудови секвенційного дерева для випадку секвенційних числень реномінативних неокласичних логік цілком аналогічна відповідній процедурі для випадку пропозиційних секвенційних числень. При цьому враховується наявність нових правил виведення.

Для секвенційних числень реномінативних неокласичних логік справджуються теореми коректності та повноти [13].

3.2 Доказова геометрія

Геометрія взяла свій початок у 3 тисячолітті до н.е. на території Вавилонії та Індській цивілізації. Первісна геометрія була набором невпорядкованих емпіричних понять про довжину, площу та об'єми, які були розроблені для задоволення потреб картографії, астрономії та будівельної справи. Найбільший прорив відбувся у древньогрецьких математиків. Для них геометрія була

найвидатнішим досягненням їхньої науки, досягнувши повноти, якої не змогла досягнути жодна інша галузь знань. Вони формально визначили велику кількість фігур, кривих, поверхонь та об’ємних тіл, змінили методологію від методу спроб та помилок до дедуктивного методу та почали використовувати аксіоматичний підхід, який і досі широко застосовується.

Першою формальною працею з використанням аксіоматичного підходу є «Начала» Евкліда. Проте постулати з цього трактату не були достатніми для опису геометрії. Перша повна система аксіом була запропонована Д. Гільбертом у кінці XIX століття [4].

Процес виведення починається з певних початкових точок. Ці початкові точки це неозначувані поняття та аксіоми. Неозначувані поняття – це базове поняття, визначення якого не дається. В геометрії це точки, лінії та площини, а також поняття перетину двох об’єктів. Аксіома – це твердження про ці поняття. Припускається, але не доводиться, що аксіоми правдиві [5].

РОЗДІЛ 4 СУТНІСТЬ МЕТОДУ

4.1 Формалізація геометричних фактів

Розглянемо твердження «фігура ABCD є квадратом». Будемо позначати його як $\square\ ABCD$. Загальновідомі твердження про квадрат включають:

Властивості:

Квадрат є прямокутником (Еквівілентне твердження: якщо фігура – квадрат, то вона є прямокутником):

$$\square\ ABCD \Rightarrow \square ABCD$$

Квадрат є ромбом (Якщо ABCD – квадрат, то ABCD – ромб):

$$\square\ ABCD \Rightarrow \diamond ABCD$$

Оскільки всі властивості мають виконуватись одночасно, то можна стверджувати:

$$\square\ ABCD \Rightarrow \square\ ABCD \wedge \diamond ABCD$$

Ознаки:

Якщо в прямокутника суміжні сторони рівні – то це квадрат (Якщо фігура одночасно є прямокутником і має рівні суміжні сторони – то це квадрат):

$$\square ABCD \wedge (AB = BC) \Rightarrow \square\ ABCD$$

Якщо в ромба є прямий кут – то це квадрат (Якщо фігура одночасно є ромбом і має прямий кут – то це квадрат):

$$\lozenge\ ABCD \wedge \left(\angle ABC = \frac{\pi}{2}\right) \Rightarrow \square\ ABCD$$

Оскільки достатньо виконання будь-якої ознаки, то:

$$(\square ABCD \wedge (AB = BC)) \vee \left(\lozenge\ ABCD \wedge \left(\angle ABC = \frac{\pi}{2}\right)\right) \Rightarrow \square\ ABCD$$

Загалом, будь-яка властивість фігури має форму: «якщо X – фігура з множини A, то виконується твердження $P(X)$»:

$$X \in A \Rightarrow P(X)$$

а ознаки мають вигляд: «якщо виконується твердження $S(X)$, то X належить множині А»:

$$S(X) \Rightarrow X \in A$$

Таким чином якщо $P = \{P_1, P_2, \dots\}$ – множина властивостей, $S = \{S_1, S_2, \dots\}$ – множина ознак X, то

$$X \in A \Rightarrow \bigcap P$$
$$\bigcup S \Rightarrow X \in A$$

На основі цих тверджень можна сформулювати правила виведення в секвенційному численні:

$$\left[\bigcap P_i, \Gamma \Rightarrow \Delta\right] \vdash [X \in A, \Gamma \Rightarrow \Delta]$$
$$\left[\Gamma \Rightarrow \Delta, \bigcup S_i\right] \vdash [\Gamma \Rightarrow \Delta, X \in A]$$

В програмній реалізації кількість елементів множин P та S залежить лише від потреб користувача і можливостей розробника. Оскільки секвенційне числення – повне, то будь-який геометричний факт можна довести маючи достатній набір програмно реалізованих правил. Також варто зазначити, що занадто велика кількість збільшує час та об’єм системних ресурсів, які використовує програма. Маловірогідно, що факти про сферу будуть потрібні при розв’язанні задачі про трикутники.

Для цієї предметної області базові формули – це рівності та нерівності з геометричними величинами (довжина відрізка, міра кута, площа тощо). В свою чергу вони є функціями вигляду $P: {}^{V}R \to \{T, F\}$, тобто V-квазіарними предикатами на множині дійсних чисел, де V – певна множина допустимих імен геометричних

величин. Вони можуть бути виведені за допомогою правил виведення в якості властивостей або ознак певних геометричних об'єктів. В результаті перетворень буде отримано секвенцію:

$$[\Gamma_1(x_1, x_2, \dots), \Gamma_2(x_1, x_2, \dots), \dots \Rightarrow \Delta_1(x_1, x_2, \dots), \Delta_2(x_1, x_2, \dots), \dots]$$

Ліва частина є системою рівнянь та нерівностей, права є набором умов. Секвенція буде виконуватись якщо кожен розв'язок $\{x_1 = X_1, x_2 = X_2, \dots\}$ системи

$$\begin{cases} \Gamma_1(x_1, x_2, \dots) \\ \Gamma_2(x_1, x_2, \dots) \\ \dots \end{cases}$$

буде задовольняти хоча б одній із умов

$$\begin{bmatrix} \Delta_1(x_1, x_2, \dots) \\ \Delta_2(x_1, x_2, \dots) \\ \dots \end{bmatrix}$$

Якщо система рівнянь несумісна – то секвенція виконується завжди (з протиріччя можна вивести що завгодно).

Хоч правила виведення секвенційного числення незастосовні до базових формул, аналогічно до того як людина розв'язує системи рівнянь з багатьма змінними, можна працювати і з такими секвенціями.

Одним з таких методів є метод підстановки. В його основі лежить факт, що розв'язок певного рівняння із системи за однією змінною можна підставити в усі інші рівняння. В результаті такого перетворення множина розв'язків системи не зміниться, проте фактична кількість невідомих буде зменшена [12]. Цей алгоритм можна записати формально:

Алгоритм «Метод підстановки»

Дано:

Система S рівнянь $\{E_1, E_2, \dots\}$, з невідомими величинами $\{x_1, x_2, \dots\}$

Початок:

Обрати рівняння E_i

Обрати змінну x_k

Знайти множину розв'язків рівняння E_i відносно змінної x_k: $X_k = \{x_k = x_{k1}, \dots\}$

Створити порожню множину розв'язків X

Для кожного розв'язку X_{km} із множини X_k:

Утворити нову систему $S_m = R_{X_{km}}^{x_k} S$

Застосувати алгоритм «Розв'язання системи» до системи S_m

Додати отримані розв'язки до множини X

Повернути множину X

Кінець

Слід зазначити що людина, як правило, не дотримується його строго. Наприклад, під час вибору рівнянь перевага буде віддаватися простішим для розв'язання з власного досвіду (наприклад лінійним над тригонометричними). Проте для програми розрізнення рівнянь за складністю не є тривіальною задачею. Окрім того на її виконання буде затрачено ресурси середовища. Тому доречно виконувати певний вид перебору розв'язків. Іншою проблемою є відсутність загального алгоритму для знаходження точних розв'язків довільного рівняння (а на практиці труднощі з їх реалізацією). В такому разі, результуюча програма повинна вміти обробляти такі випадки. Потрібно також відрізняти відсутність розв'язків рівняння від неможливості їх знаходження наявними засобами.

4.2 Основна ідея

Оскільки операція реномінації визначена для предикатів:

$$R_X^{x_n}\big(P(x_1, x_2, \dots, x_{n-1}, x_n, x_{n+1}, \dots)\big) \stackrel{\text{def}}{=} P(x_1, x_2, \dots, x_{n-1}, X, x_{n+1}, \dots)$$

то для зручності можна визначити її і для множин предикатів:

$$R_X^x\{P_1, P_2, \dots\} \stackrel{\text{def}}{=} \{R_X^x P_1, R_X^x P_2, \dots\}$$

та для секвенцій:

$$R_X^x[\Gamma \Rightarrow \Delta] \stackrel{\text{def}}{=} [R_X^x \Gamma \Rightarrow R_X^x \Delta]$$

Програмне доведення секвенцій шляхом пошуку розв'язку системи рівнянь та нерівностей (предикатів) буде виглядати таким чином:

1) Для кожного рівняння знаходиться множина його розв'язків X_i відносно кожної змінної x_i (область істиності предиката). Кожна така множина є одним із можливих шляхів пошуку розв'язку системи.

2) Для кожної множини секвенція буде виконуватись, якщо при підстановці кожного розв'язку $X_{i,j}$ у секвенцію (реномінації $R^{x_i}_{X_{i,j}}(S)$), результуючі секвенції також будуть виконуватись: $S = R^{x_i}_{X_{i,1}}(S) \wedge R^{x_i}_{X_{i,2}}(S) \wedge \ldots$

3) Оскільки всі шляхи розв'язання приводять до одного і того самого результату, то всі результуючі набори секвенцій можна об'єднати диз'юнкцією.

Через особливість програмної реалізації для секвенцій будуть розглядатися 3 можливі значення: «виконується», «не виконується», «відповіді немає». Такі значення можна зручно описати через тризначну логіку (правда, неправда, невідомо). Очевидно, що дві системи рівнянь, які були отримані з вхідної через розв'язання двох різних рівнянь, або одного рівняння відносно двох різних змінних будуть мати однакову множину розв'язків. Враховуючи цей факт, можемо записати результат як:

$$\left(R^{x_1}_{X_{1,1}}(S) \wedge R^{x_1}_{X_{1,2}}(S) \wedge \ldots\right) \vee \left(R^{x_2}_{X_{2,1}}(S) \wedge R^{x_2}_{X_{2,2}}(S) \wedge \ldots\right) \vee \ldots$$

Якщо якийсь із диз'юнктів не може бути обчислений («відповіді немає»), то його можна прибрати з диз'юнкції, оскільки на результат він не впливає ($x \vee U = x$). Для уникнення рекурсії має сенс подати вказаний вираз у вигляді кон'юнкції (щоб його можна було доводити як звичайне секвенційне дерево).

$$\left(R^{x_1}_{X_{1,1}}(S) \vee R^{x_2}_{X_{2,1}}(S) \vee \ldots\right) \wedge \left(R^{x_1}_{X_{1,1}}(S) \vee R^{x_2}_{X_{2,2}}(S) \vee \ldots\right) \wedge \ldots$$

Враховуючи особливості секвенцій(ліві та праві частини диз'юнктів виконуються одночасно) можна подати диз'юнкцію двох секвенцій у вигляді однієї:

$$[\Gamma_1, \Gamma_2, \ldots \Rightarrow \Delta_1, \Delta_2, \ldots] \vee [\Phi_1, \Phi_2, \ldots \Rightarrow \Psi_1, \Psi_2, \ldots]$$

$$[\Gamma_1, \Gamma_2, \ldots, \Phi_1, \Phi_2, \ldots \Rightarrow \Delta_1, \Delta_2, \ldots, \Psi_1, \Psi_2, \ldots]$$

Тому вирази в дужках можливо спростити. В результаті цих перетворень з однієї секвенції вийшла кон’юнкція секвенцій. Аналогічний результат дають і правила виведення секвенційного числення. Відмінностями є кількість розгалужень та той факт, що правило застосовується на всю секвенцію в цілому, а не тільки на одну формулу.

У випадку неможливості розв’язання системи за допомогою реалізованих програмно способів висновок про виконуваність секвенції буде невизначений.

РОЗДІЛ 5 ПРОЄКТУВАННЯ ПРОГРАМИ

5.1 SOLID

Під час розробки даного програмного продукту були дотримані принципи SOLID. Відносно розробки програмного забезпечення, SOLID – це абревіатура для 5 принципів проєктування, покликаних зробити об’єктно-орієнтовані продукти більш зрозумілими, гнучкими та з широкими можливостями для підтримки. Одночасне використання цих принципів було запропоноване американським розробником Р. Мартіном у його роботі присвяченій висвітленню проблеми «гниття» програмного забезпечення. Основними ідеями є:

– Принцип єдиної відповідальності (single-responsibility principle) – це принцип в програмуванні, який стверджує що «Кожен модуль має бути відповідальним за одну і лише одну функцію». Засновник терміну описує його як «Клас може мати лише одну причину для зміни»

– Принцип відкритості-закритості (open–closed principle) стверджує що сутність має бути відкритою до розштирення, проте закритою для внесення змін.

– Принцип підстановки Лісков (Liskov substitution principle) стверджує що об’єкт має бути замінним на його нащадка зі збереженням правильності програми. Він покликаний гарантувати взаємозамінність типів у ієрархії, насамперед типів об’єктів.

– Принцип розділення інтерфейсу (interface segregation principle) стверджує що код не повинен залежати від методів, які він не використовує. Він розподіляє великі інтерфейси на дрібніші для того, щоб клієнти мали знати лише про ті, які представляють для них інтерес. Такі «стиснені» інтерфейси називаються рольовими.

– Принцип інверсії залежностей (dependency inversion principle) – це специфічна методологія, умовою виконання якої є незалежність абстракцій від реалізацій та модулів верхнього рівня від деталей реалізації своїх підмодулів [7].

5.2 Алгоритми

На етапі проєктування програми було формально описано основні алгоритми для її роботи.

Алгоритм «Розв’язання системи»

Дано:

Секвенція $[\Gamma \Rightarrow \Delta]$

Початок:

Якщо Перетин Γ та Δ непорожній або Γ містить хибний факт або Δ містить істинний:

Позначити секвенцію як замкнену

Повернути секвенцію

Якщо Γ містить істинний факт або Δ містить хибний:

Видалити факт

Повернути секвенцію

Якщо Γ порожня або Δ порожня:

Позначити секвенцію як незамкнену

Повернути секвенцію

Інакше:

Створити набір секвенцій $S = \{[\Rightarrow]\}$

Для кожного елемента E із Γ:

Для кожної змінної x із E:

Застосувати алгоритм «Розв’язання» для E та x

Якщо результат визначений:

Для кожного розв’язку $x = X$:

Обчислити $R = [R_X^x \Gamma \Rightarrow R_X^x \Delta]$

Створити порожній набір секвенцій N

Для кожної секвенції $[\Phi \Rightarrow \Psi]$ із S:

Додати $[\Phi \Rightarrow \Psi] \vee R$ в N

Прирівняти S до N

Якщо жоден результат не був визначений:

Повернути невизначеність

Інакше:

Повернути S

Кінець

Важливо щоб програма могла завжди завершитись у скінченний час. При підстановці у секвенцію $[\Gamma \Rightarrow \Delta]$ будемо залишати в множині Γ тільки результати тих підстановок, які зменшують кількість невідомих у виразі.

Алгоритм «Правило виведення»

Дано:

Секвенція $[\Gamma \Rightarrow \Delta]$

Початок:

Якщо Δ порожня:

Позначити секвенцію як незамкнену

Повернути секвенцію

Якщо Перетин Γ та Δ непорожній:

Позначити секвенцію як замкнену

Повернути секвенцію

Інакше:

Обрати пропозиційну зв’язку із Γ або Δ

Застосувати відповідне правило виведення

Повернути результат

Кінець

При розгалуженні, наприклад у випадку $[A \vee B, \Gamma \Rightarrow \Delta]$, алгоритм буде повертати набір секвенцій, наприклад $\{[A, \Gamma \Rightarrow \Delta], [B, \Gamma \Rightarrow \Delta]\}$. Це необхідно для виконання алгоритму пошуку в ширину.

Алгоритм «Побудова дерева»

Дано:

Набір секвенцій $S = \{[\Gamma_1 \Rightarrow \Delta_1], [\Gamma_2 \Rightarrow \Delta_2], \dots\}$

Початок:

Повторювати доки набір непорожній:

Створити порожній набір секвенцій N

Для кожної секвенції $[\Gamma_i \Rightarrow \Delta_i]$ з набору:

Вилучити секвенцію з набору

Якщо $[\Gamma_i \Rightarrow \Delta_i]$ замкнена:

Вивести «$[\Gamma_i \Rightarrow \Delta_i]$ замкнена»

Якщо $[\Gamma_i \Rightarrow \Delta_i]$ незамкнена:

Вивести «$[\Gamma_i \Rightarrow \Delta_i]$ незамкнена»

Повернути неправда

Інакше якщо $[\Gamma_i \Rightarrow \Delta_i]$ містить пропозиційні зв'язки:

Застосувати алгоритм «Правило виведення» до $[\Gamma_i \Rightarrow \Delta_i]$

Додати результат до набору N

Інакше:

Застосувати алгоритм «Розв'язання системи» до $[\Gamma_i \Rightarrow \Delta_i]$

Якщо результат невизначеність:

Вивести «Неможливо знайти розв'язки $[\Gamma_i \Rightarrow \Delta_i]$»

Повернути невизначеність

Інакше:

Додати результат до набору N

Додати секвенції з набору N в набір S

Повернути правда

Кінець

5.3 Архітектура програми

На етапі розробки було створено таку структуру класів. Завдяки використанням приницпів SOLID при написанні коду програму можна легко доповнити додатковими частинами за потреби. Детальні діаграми класів розміщено в додатку А.

`package algebra` – містить класи алгебраїчних виразів. (Рисунок .1)

`AlgebraicExpression.class` – абстрактний клас для представлення алгебраїчного виразу та роботи з ним: арифметичні дії, порівняння, рядкове представлення, хеш-код, заміна змінних іншими виразами.

`package function` – містить класи алгебраїчних функцій.

`AlgebraicExpressionFunction.class` – абстрактний клас для представлення функції одного аргументу в якості частини виразу.

`AlgebraicExpressionSqrt.class` – клас для представлення функції арифметичного квадратного кореня.

`AlgebraicExpressionSin.class` – аналогічно до попереднього, функція синуса кута.

`AlgebraicExpressionCos.class` – …

`AlgebraicExpressionAcos.class` – …

… – за потреби можна додати інші класи функцій.

`AlgebraicExpressionSequence.class` – абстрактний клас для представлення певної послідовності елементів у виразі та роботи з нею: додавання нового члена послідовності.

`AlgebraicExpressionSum.class` – клас для представлення суми довільної кількості доданків у виразі.

`AlgebraicExpressionProduct.class` – клас для представлення добутку довільної кількості множників у виразі.

`AlgebraicExpressionPower.class` – клас для представлення цілого степеня виразу.

`AlgebraicExpressionNumber.class` – клас для представлення дійсного числа у виразі.

`AlgebraicExpressionVariable.class` – клас для представлення алгебраїчної змінної у виразі.

`AlgebraicExpressionConstant.class` – клас для представлення математичної константи, наприклад числа пі або числа Ейлера, у виразі.

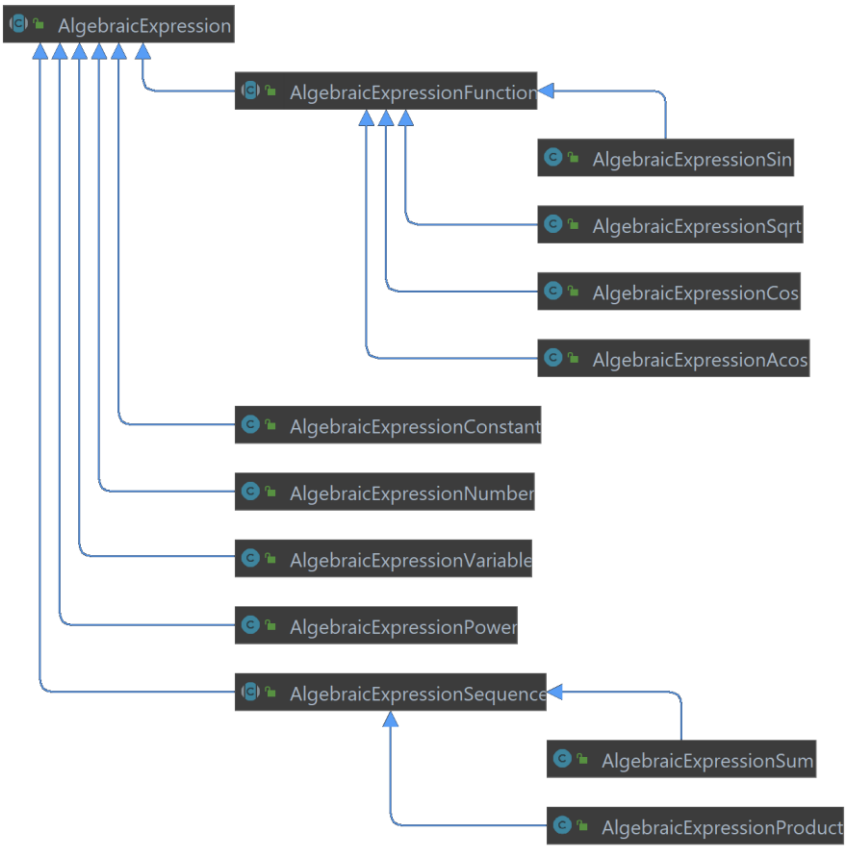


Рисунок 5.1 – схема класів програми, `package algebra`.

`package logic` – містить класи логічних виразів. (Рисунок .2)

`BooleanExpression.class` – клас для представлення цілого степеня виразу.

`package geometry` – містить класи геометричних фактів в якості логічних виразів.

… – за потреби можна додати інші класи фігур та відношень.

`BooleanExpressionBinaryOperation.class` – абстрактний клас для представлення логічної операції над двома аргументами.

`BooleanExpressionAnd.class` – клас для представлення кон’юнкції двох аргументів.

`BooleanExpressionOr.class` – аналогічно до попереднього, диз'юнкція.

`BooleanExpressionImply.class` – …

`BooleanExpressionEquals.class` – …

… – за потреби можна додати інші класи логічних операцій.

`BooleanExpressionNot.class` – клас для представлення заперечення.

`BooleanExpressionRenomination.class` – клас для представлення операції реномінації.

`BooleanExpressionAtom.class` – клас для представлення базової формули, яка є умовою від декількох змінних. Має вигляд рівняння або нерівності.

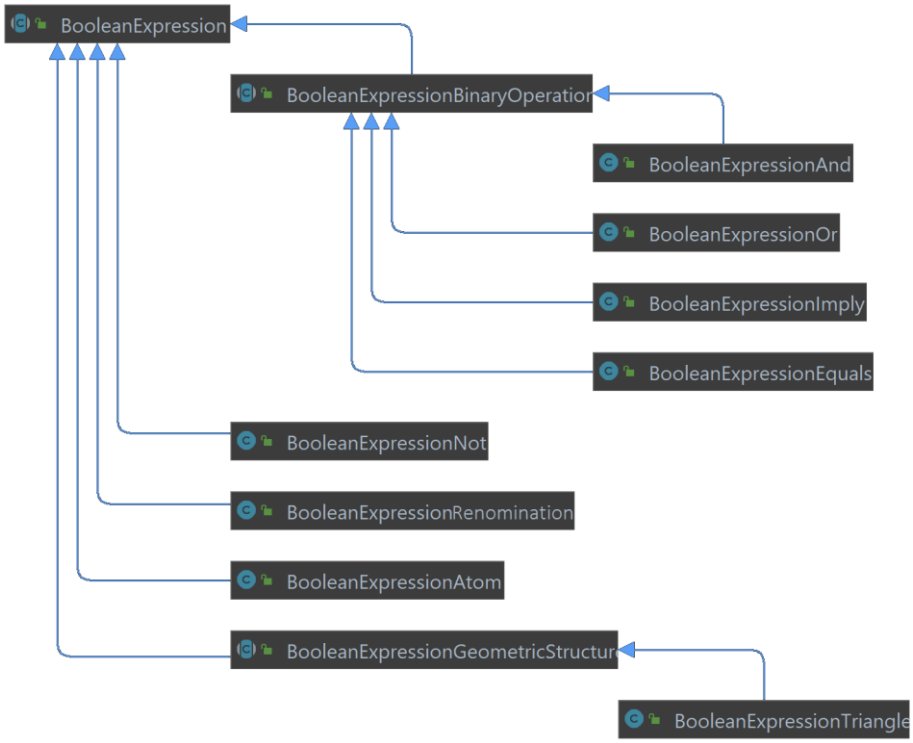


Рисунок 5.2 – схема класів програми, `package logic`.

`package structure` – містить класи для різних математичних абстракцій та об'єктів, які використовуються для обчислень.

`Polynomial.class` – клас для представлення многочлена від однієї змінної. Приведення виразів до форми многочлена використовується при розв'язанні рівнянь.

`Solvable.class` – абстрактний клас для представлення певної умови. Основними методами є пошук значень, які задовольняють умову та перевірка їх коректності.

`Equation.class` – клас для представлення рівняння.

`Equation.class` – клас для представлення нерівності.

`Sequent.class` – клас для представлення секвенції. Містить методи для доведення секвенцій та виконання перетворень згідно з правилами виведення. Містить приватні поля множин формул та змінних лівої та правої частини, методи, які визначають чи є секвенція термінальною і чи замкнена вона. Замість збереження усього секвенційного дерева програма працює лише з набором його нетермінальних листків, що довзоляє підвищити продуктивність та зменшити кількість затрачених системних ресурсів

`Solution.class` – клас для представлення конкретного розв'язку рівняння або нерівності. Містить методи для підстановки отриманих значень у вирази.

`Solutions.class` – абстрактний клас для представлення множини розв'язків рівняння або нерівності. Містить методи для роботи з такими множинами, наприклад об'єднання та перетин множин, перевірка чи задовольняють розв'язки довільну умову.

`SolutionsAll.class` – клас для представлення факту, що будь-який розв'язок задовольняє умові.

`SolutionsAll.class` – клас для представлення факту, що рівняння або нерівність розв'язків не мають.

`SolutionsSet.class` – клас для представлення скінченного набору розв'язків.

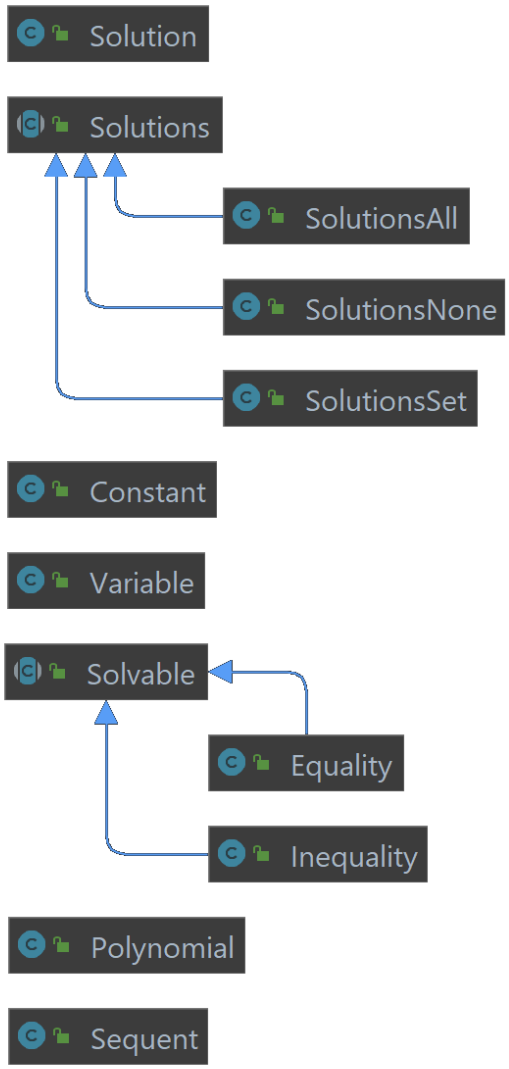


Рисунок 5.3 – схема класів програми, `package structure`.

РОЗДІЛ 6 ТЕСТУВАННЯ

Після завершення розробки програми було проведене її тестування. Розглянемо формулу $(\neg B \vee C \rightarrow \neg(\neg A \vee B) \vee (\neg A \vee C))$. Побудувавши таблицю істинності можна переконатися, що цей вираз є тавтологією. Тепер розглянемо роботу підпрограми для доведення секвенції, проаналізувавши результат, який виводиться при введенні $[\Rightarrow (\neg B \vee C \rightarrow \neg(\neg A \vee B) \vee (\neg A \vee C))]$:

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=56569:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
D:\JavaProjects\Kursova\out\production\Kursova main.Main
!B|C>!(!A|B)|(!A|C)
{ { ⊣¬BVC→¬(¬AVB)V¬AVC } }
{ ⊣¬BVC→¬(¬AVB)V¬AVC }: За ПВ8 маємо: { ⊢¬BVC, ⊣¬(¬AVB)V¬AVC }
{ ⊢¬BVC, ⊣¬(¬AVB)V¬AVC }: За ПВ3 маємо: { ⊢C, ⊣¬(¬AVB)V¬AVC }, {
⊢¬B, ⊣¬(¬AVB)V¬AVC }
{ ⊢C, ⊣¬(¬AVB)V¬AVC }: C є пропозиційною змінною.
{ ⊢¬B, ⊣¬(¬AVB)V¬AVC }: За ПВ1 маємо: { ⊣¬(¬AVB)V¬AVC, ⊣B }
{ ⊣¬(¬AVB)V¬AVC, ⊢C }: За ПВ4 маємо: { ⊣¬AVC, ⊣¬(¬AVB), ⊢C }
{ ⊣¬(¬AVB)V¬AVC, ⊣B }: За ПВ4 маємо: { ⊣¬AVC, ⊣¬(¬AVB), ⊣B }
{ ⊣¬AVC, ⊣¬(¬AVB), ⊢C }: За ПВ4 маємо: { ⊣¬(¬AVB), ⊣C, ⊣¬A, ⊢C }
{ ⊣¬AVC, ⊣¬(¬AVB), ⊣B }: За ПВ4 маємо: { ⊣¬(¬AVB), ⊣B, ⊣C, ⊣¬A }
{ ⊣¬(¬AVB), ⊣C, ⊣¬A, ⊢C }: За ПВ2 маємо: { ⊢¬AVB, ⊣C, ⊣¬A, ⊢C }
{ ⊣¬(¬AVB), ⊣B, ⊣C, ⊣¬A }: За ПВ2 маємо: { ⊢¬AVB, ⊣B, ⊣C, ⊣¬A }
{ ⊢¬AVB, ⊣C, ⊣¬A, ⊢C }: За ПВ3 маємо: { ⊢B, ⊣C, ⊣¬A, ⊢C }, { ⊢¬A,
⊣C, ⊣¬A, ⊢C }
{ ⊢¬AVB, ⊣B, ⊣C, ⊣¬A }: За ПВ3 маємо: { ⊢B, ⊣B, ⊣C, ⊣¬A }, { ⊢¬A,
⊣B, ⊣C, ⊣¬A }
{ ⊢B, ⊣C, ⊣¬A, ⊢C }: B є пропозиційною змінною.
{ ⊢¬A, ⊣C, ⊣¬A, ⊢C }: За ПВ1 маємо: { ⊣A, ⊣C, ⊣¬A, ⊢C }
```

```
{ ⊢B, ⊣B, ⊣C, ⊣¬A }: B є пропозиційною змінною.
{ ⊢¬A, ⊣B, ⊣C, ⊣¬A }: За ПВ1 маємо: { ⊣A, ⊣B, ⊣C, ⊣¬A }
{ ⊣C, ⊣¬A, ⊢B, ⊢C }: C є пропозиційною змінною.
{ ⊣A, ⊣C, ⊣¬A, ⊢C }: A є пропозиційною змінною.
{ ⊣B, ⊣C, ⊣¬A, ⊢B }: B є пропозиційною змінною.
{ ⊣A, ⊣B, ⊣C, ⊣¬A }: A є пропозиційною змінною.
{ ⊣¬A, ⊢B, ⊢C, ⊣C }: За ПВ2 маємо: { ⊢A, ⊢B, ⊢C, ⊣C }
{ ⊣C, ⊣¬A, ⊢C, ⊣A }: C є пропозиційною змінною.
{ ⊣C, ⊣¬A, ⊢B, ⊣B }: C є пропозиційною змінною.
{ ⊣B, ⊣C, ⊣¬A, ⊣A }: B є пропозиційною змінною.
{ ⊢A, ⊢B, ⊢C, ⊣C }: A є пропозиційною змінною.
{ ⊣¬A, ⊢C, ⊣A, ⊣C }: За ПВ2 маємо: { ⊢A, ⊢C, ⊣A, ⊣C }
{ ⊣¬A, ⊢B, ⊣B, ⊣C }: За ПВ2 маємо: { ⊢A, ⊢B, ⊣B, ⊣C }
{ ⊣C, ⊣¬A, ⊣A, ⊣B }: C є пропозиційною змінною.
{ ⊢A, ⊢B, ⊢C, ⊣C }: не містить пропозиційних зв'язок. Секвенція
замкнена.
{ ⊢A, ⊢C, ⊣A, ⊣C }: A є пропозиційною змінною.
{ ⊢A, ⊢B, ⊣B, ⊣C }: A є пропозиційною змінною.
{ ⊣¬A, ⊣A, ⊣B, ⊣C }: За ПВ2 маємо: { ⊢A, ⊣A, ⊣B, ⊣C }
{ ⊢A, ⊢C, ⊣A, ⊣C }: не містить пропозиційних зв'язок. Секвенція
замкнена.
{ ⊢A, ⊢B, ⊣B, ⊣C }: не містить пропозиційних зв'язок. Секвенція
замкнена.
{ ⊢A, ⊣A, ⊣B, ⊣C }: A є пропозиційною змінною.
{ ⊢A, ⊣A, ⊣B, ⊣C }: не містить пропозиційних зв'язок. Секвенція
замкнена.
Секвенційне дерево термінальне і замкнене. Секвенція виконується.

Process finished with exit code 0
```

Побудувавши термінальне секвенційне дерево програма дійшла до висновку, що воно замкнене, а отже і початкова секвенція виконується.

Тепер розглянемо інший приклад $(A \to C) \to (B \vee C \to A \vee B \to C)$. Побудувавши таблицю істинності можна помітити, що вираз не є всюди істинним. Розглянемо виведення підпрограми:

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=56601:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
D:\JavaProjects\Kursova\out\production\Kursova main.Main
(A>C)>(B|C>(A|B>C))
{ { ⊣(A→C)→BVC→AVB→C } }
{ ⊣(A→C)→BVC→AVB→C }: За ПВ8 маємо: { ⊢A→C, ⊣BVC→AVB→C }
{ ⊢A→C, ⊣BVC→AVB→C }: За ПВ7 маємо: { ⊢C, ⊣BVC→AVB→C }, { ⊣A,
⊣BVC→AVB→C }
{ ⊢C, ⊣BVC→AVB→C }: C є пропозиційною змінною.
{ ⊣A, ⊣BVC→AVB→C }: A є пропозиційною змінною.
{ ⊣BVC→AVB→C, ⊢C }: За ПВ8 маємо: { ⊢BVC, ⊣AVB→C, ⊢C }
{ ⊣BVC→AVB→C, ⊣A }: За ПВ8 маємо: { ⊢BVC, ⊣AVB→C, ⊣A }
{ ⊢BVC, ⊣AVB→C, ⊢C }: За ПВ3 маємо: { ⊢C, ⊣AVB→C, ⊢C }, { ⊢B,
⊣AVB→C, ⊢C }
{ ⊢BVC, ⊣AVB→C, ⊣A }: За ПВ3 маємо: { ⊢C, ⊣AVB→C, ⊣A }, { ⊢B,
⊣AVB→C, ⊣A }
{ ⊢C, ⊣AVB→C, ⊢C }: C є пропозиційною змінною.
{ ⊢B, ⊣AVB→C, ⊢C }: B є пропозиційною змінною.
{ ⊢C, ⊣AVB→C, ⊣A }: C є пропозиційною змінною.
{ ⊢B, ⊣AVB→C, ⊣A }: B є пропозиційною змінною.
{ ⊣AVB→C, ⊢C }: За ПВ8 маємо: { ⊢AVB, ⊣C, ⊢C }
{ ⊣AVB→C, ⊢B, ⊢C }: За ПВ8 маємо: { ⊢AVB, ⊣C, ⊢B, ⊢C }
{ ⊣AVB→C, ⊢C, ⊣A }: За ПВ8 маємо: { ⊢AVB, ⊣C, ⊢C, ⊣A }
{ ⊣AVB→C, ⊢B, ⊣A }: За ПВ8 маємо: { ⊢AVB, ⊣C, ⊢B, ⊣A }
{ ⊢AVB, ⊣C, ⊢C }: За ПВ3 маємо: { ⊢B, ⊣C, ⊢C }, { ⊢A, ⊣C, ⊢C }
```

{ ⊢AVB, ⊣C, ⊢B, ⊢C }: За ПВЗ маємо: { ⊢B, ⊣C, ⊢B, ⊢C }, { ⊢A, ⊣C, ⊢B, ⊢C }

{ ⊢AVB, ⊣C, ⊢B, ⊣A }: За ПВЗ маємо: { ⊢B, ⊣C, ⊢B, ⊣A }, { ⊢A, ⊣C, ⊢B, ⊣A }

{ ⊢B, ⊣C, ⊢C }: B є пропозиційною змінною.

{ ⊢A, ⊣C, ⊢C }: A є пропозиційною змінною.

{ ⊢B, ⊣C, ⊢B, ⊢C }: B є пропозиційною змінною.

{ ⊢A, ⊣C, ⊢B, ⊢C }: A є пропозиційною змінною.

{ ⊣C, ⊢B, ⊢C }: C є пропозиційною змінною.

{ ⊣C, ⊢A, ⊢C }: C є пропозиційною змінною.

{ ⊣C, ⊢A, ⊢B, ⊢C }: C є пропозиційною змінною.

{ ⊢B, ⊢C, ⊣C }: не містить пропозиційних зв'язок. Секвенція замкнена.

{ ⊢A, ⊢C, ⊣C }: не містить пропозиційних зв'язок. Секвенція замкнена.

{ ⊢A, ⊢B, ⊢C, ⊣C }: не містить пропозиційних зв'язок. Секвенція замкнена.

{ ⊢A, ⊣C, ⊢B, ⊣A }: A є пропозиційною змінною.

{ ⊣C, ⊢A, ⊢B, ⊣A }: C є пропозиційною змінною.

{ ⊢A, ⊢B, ⊣A, ⊣C }: не містить пропозиційних зв'язок. Секвенція замкнена.

{ ⊢B, ⊣C, ⊢B, ⊣A }: B є пропозиційною змінною.

{ ⊣C, ⊢B, ⊣A }: C є пропозиційною змінною.

{ ⊢B, ⊣A, ⊣C }: не містить пропозиційних зв'язок. Секвенція незамкнена.

{ ⊢AVB, ⊣C, ⊢C, ⊣A }: За ПВЗ маємо: { ⊢B, ⊣C, ⊢C, ⊣A }, { ⊢A, ⊣C, ⊢C, ⊣A }

{ ⊢B, ⊣C, ⊢C, ⊣A }: B є пропозиційною змінною.

{ ⊣C, ⊢B, ⊢C, ⊣A }: C є пропозиційною змінною.

{ ⊢B, ⊢C, ⊣A, ⊣C }: не містить пропозиційних зв'язок. Секвенція замкнена.

{ ⊢A, ⊣C, ⊢C, ⊣A }: A є пропозиційною змінною.

```
{ ⊣C, ⊢A, ⊢C, ⊣A }: C є пропозиційною змінною.
{ ⊢A, ⊢C, ⊣A, ⊣C }: не містить пропозиційних зв'язок. Секвенція
замкнена.
Секвенційне дерево термінальне і незамкнене. Секвенція не
виконується.

Process finished with exit code 0
```

Як бачимо програма вказала, що секвенційне дерево є незамкненим, а отже початкова формула не є тавтологією. Також можна помітити, що незамкнений листок відповідає інтерпретації (0,1,0), при якій формула є хибною. Результат роботи програми в обох тестових випадках збігається з правильним.

Розглянемо просту систему рівнянь:

$$\begin{cases} x^2 - y = 4 \\ x^2 - y = -4 \end{cases}$$

Розв’язання системи наведено у додатку Б:

$$\{(x = -2, y = 0), (x = 2, y = 0)\}$$

Перевіримо чи задовольняють розв’язки умову $y = 0$. Це твердження еквівалентно тому чи виконується секвенція:

$$[\vdash (x^2 - y = 4), \vdash (x^2 - y = -4), \dashv (y = 0)]$$

Розглянемо виведення програми:

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=57405:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
"C:\Users\somerandomsymbols\OneDrive\Рабочий
стол\krb\KRB\out\production\KRB" test.TestSequents
Sequents
[{ ⊢( x^2 + y ) = 4.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣y = 0.0 }]
```

... (Результати проміжних обчислень в додатку В)

```
closed - true

Process finished with exit code 0
```

Як бачимо секвенція виконується, отже розв’язки задовольняють умову. Тепер перевіримо інше твердження $x + y = 0$. Аналогічно перевіримо чи виконується секвенція:

$$[\vdash (x^2 - y = 4), \vdash (x^2 - y = -4), \dashv (x + y = 0)]$$

Розглянемо виведення програми:

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=57483:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
"C:\Users\somerandomsymbols\OneDrive\Рабочий
стол\krb\KRB\out\production\KRB" test.TestSequents
Sequents
[{ ⊢( x^2 + y ) = 4.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣( y + x ) =
0.0 }]
```

... (Результати проміжних обчислень в додатку Г)

```
closed - false

Process finished with exit code 0
```

Як бачимо секвенція незамкнена, отже умова з системи не випливає. Також можна переглянути результати проміжних перетворень і переконатись у їх правильності. Зазначимо, що тип рівнянь у системі не має жодного впливу на основну ідею програми. В разі внеможливості визначення розв’язку наявними методами, можна відносно просто розширити програму, щоб вона включала в себе ці методи.

Для систем з більшою кількістю рівнянь, навіть найпростіші геометричні задачі, програма буде виконувати велику кількість проміжних обчислень. В наслідок цього виведення програми матиме занадто великий об’єм (десятки тисяч секвенцій), тому виведення результатів проміжних обчислень вимкнено. Проте, можна перевірити остаточний результат. Візьмемо задачу:

Дано: $\triangle ABC, AB = 3, BC = 4, \measuredangle ABC = \frac{\pi}{2}$

Довести: $AC = 5$

За допомогою теореми Піфагора (або її узагальнення – теореми косинусів) можна з’ясувати довжину сторони AB і переконатися, що вона справді дорівнює п'яти.

Розглянемо результат роботи програми на вхідних даних:

$$\left[\vdash \triangle ABC, \vdash AB = 3, \vdash BC = 4, \vdash \measuredangle ABC = \frac{\pi}{2}, \dashv AC = 5\right]$$

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=57529:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
"C:\Users\somerandomsymbols\OneDrive\Рабочий
стол\krb\KRB\out\production\KRB" test.TestSequents
Sequents
[{ ⊢( ∠[A,C,B] + ( -0.5 * π ) ) = 0.0, ⊢( d[A,C] + -3.0 ) = 0.0, ⊢(
d[B,C] + -4.0 ) = 0.0, ⊢△{A; B; C}, ⊣( d[A,B] + -5.0 ) = 0.0 }]

closed - true

Process finished with exit code 0
```

Тепер розглянемо результат на суперечливих вхідних даних.

$$\left[\vdash \triangle ABC, \vdash AB = 3, \vdash BC = 4, \vdash \measuredangle ABC = \frac{\pi}{2}, \dashv AC = 6\right]$$

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-
javaagent:D:\JetBrains\IntelliJ IDEA
2022.1.2\lib\idea_rt.jar=57598:D:\JetBrains\IntelliJ IDEA
2022.1.2\bin" -Dfile.encoding=UTF-8 -classpath
"C:\Users\somerandomsymbols\OneDrive\Рабочий
стол\krb\KRB\out\production\KRB" test.TestSequents
Sequents
[{ ⊢△{A; B; C}, ⊢( -4.0 + d[B,C] ) = 0.0, ⊢( -3.0 + d[A,C] ) = 0.0,
⊢( ( π * -0.5 ) + ∠[A,C,B] ) = 0.0, ⊣( d[A,B] + -6.0 ) = 0.0 }]

closed - false

Process finished with exit code 0
```

Як видно, програма дає правильну відповідь на найпростіші запити. Враховуючи раніше доведені факти та спираючись на теорему про повноту секвенційного числення можна зробити позитивний висновок про її коректність.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено програмний продукт для автоматизації доведення геометричних тверджень.

Для реалізації цього плану було зроблено наступне:

– Проведено аналіз наявних продуктів з даної тематики на ринку і зроблено висновок, що продуктів, які поглиблено спеціалізуються на геометрії, наразі немає.

– Проведено аналіз найчастіше вживаних інструментів розробки та обрано оптимальні варіанти, які і були використані при написанні програми.

– Розроблено детальну архітектуру програмної системи, а також формально записано алгоритми потрібні для її роботи, внаслідок чого швидкість написання коду збільшилась.

– Здійснено програмну реалізацію продукту з урахуванням результатів отриманих раніше.

Розроблений продукт має такі функції:

- Перетворення набору геометричних фактів у послідовність придатну для опрацювання.
- Здійснення пошуку доведень на основі секвенційного числення.
- Можливість виведення проміжних результатів.
- Виведення результату.

Завдяки дотриманню принципів SOLID можливе доопрацювання та розширення розробленого продукту, наприклад додавання можливості для розв’язання діофантових рівнянь за допомогою оператора мінімізації. Також, за потреби, є можливість повної зміни предметної області для пошуку доведень, оскільки основна структура, де реалізоване секвенційне числення, не потребуватиме змін в такому разі. Також можливим є створення повноцінного користувацького інтерфейсу для полегшення роботи з програмою для кінцевого користувача.

В підсумку було розроблено продукт, який може використовуватись як при розробці інших програмних застосунків, так і для практичного застосування.

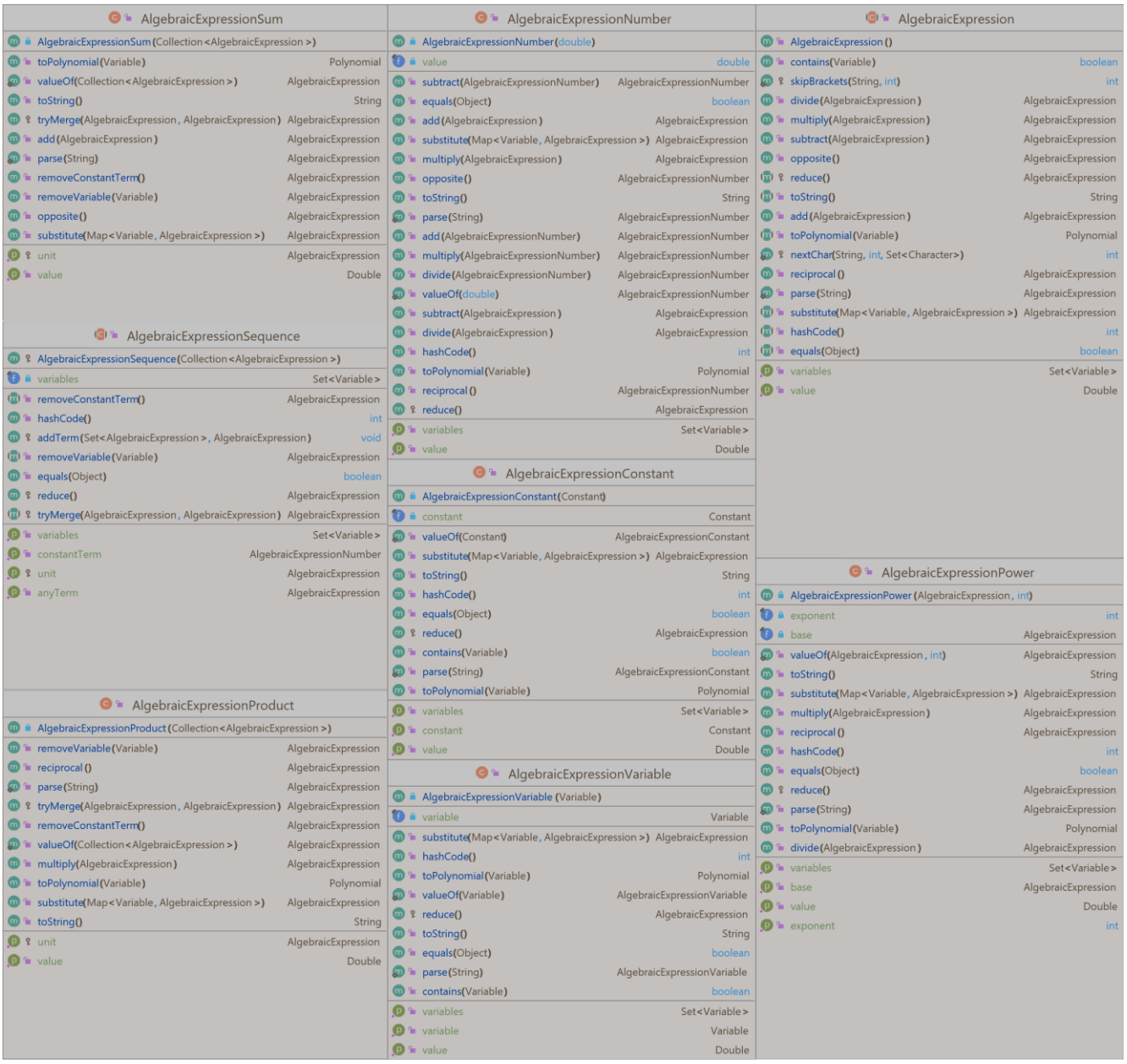
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

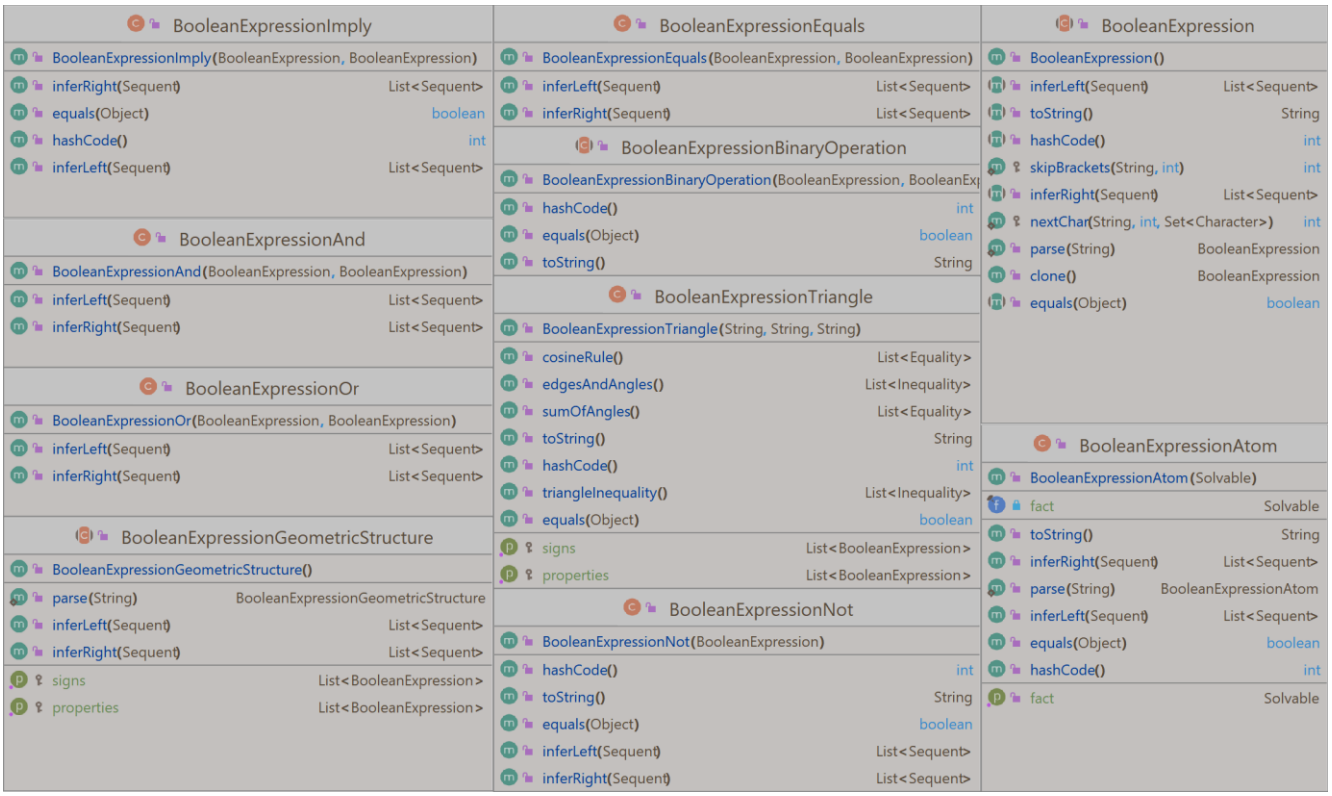
1. Зубенко В. В. Основи Математичної логіки / В. В. Зубенко, С. С. Шкільняк. – Київ: Національний університет біоресурсів і природокористування України, 2020. – 102 с.
2. Gentzen G. Untersuchungen über das logische Schließen. I / Gerhard Gentzen. // Mathematische Zeitschrift. – 1935. – №39. – С. 176–210.
3. Gentzen G. Untersuchungen über das logische Schließen. II / Gerhard Gentzen. // Mathematische Zeitschrift. – 1935. – №39. – С. 405–431.
4. Cooke R. The History of Mathematics / Roger Cooke. – New York: Wiley-Interscience, 2005. – 632 с.
5. Klein F. Elementary Mathematics from an Advanced Standpoint: Geometry / Felix Klein. – New York: Dover, 1948. – 176 с.
6. Fowler M. Crossing Refactoring's Rubicon [Електронний ресурс] / Martin Fowler. – 2001. – Режим доступу до ресурсу: https://www.martinfowler.com/articles/refactoringRubicon.html.
7. Martin R. Getting a SOLID start [Електронний ресурс] / Robert Martin. – 2009. – Режим доступу до ресурсу: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start.
8. About Symbolab [Електронний ресурс] – Режим доступу до ресурсу: https://www.symbolab.com/about.
9. About Wolfram|Alpha: Making the world's knowledge computable [Електронний ресурс] – Режим доступу до ресурсу: https://www.wolframalpha.com/about.
10. Features of Java - Javatpoint [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/features-of-java.
11. IntelliJ IDEA overview | IntelliJ IDEA Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html.

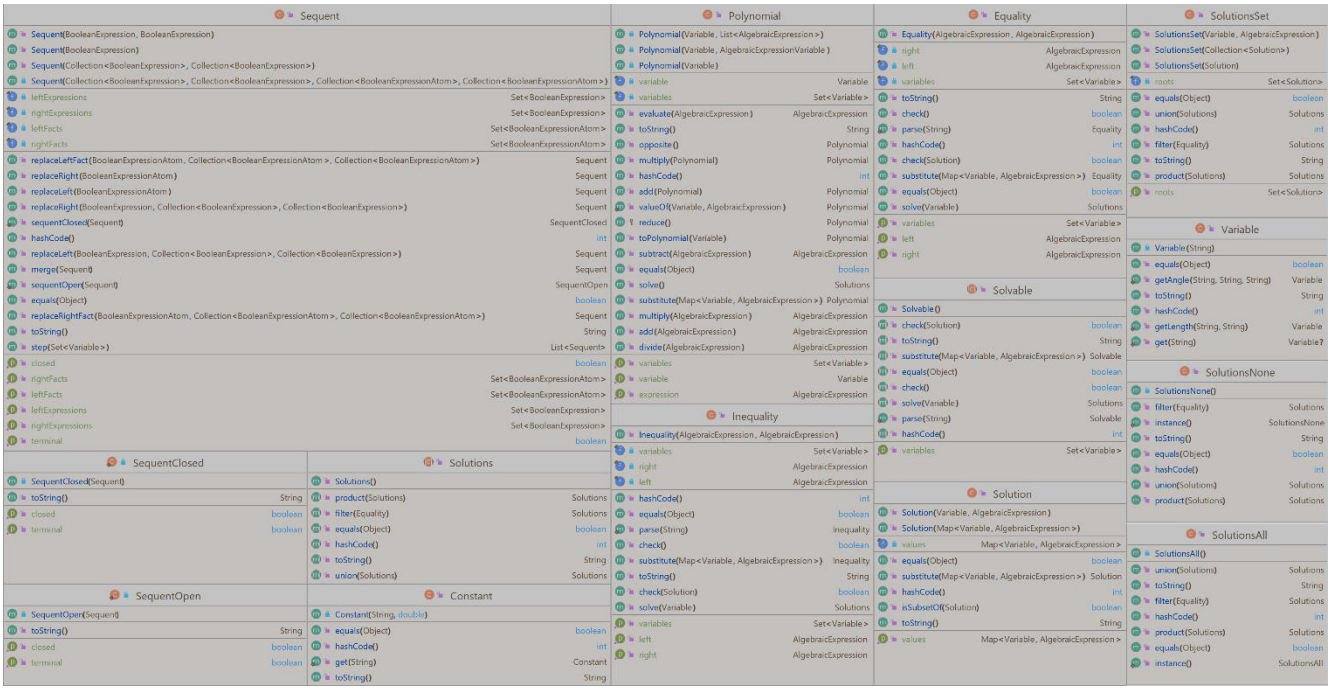
12. Systems of equations with substitution [Електронний ресурс] – Режим доступу до ресурсу: https://www.khanacademy.org/math/cc-eighth-grade-math/cc-8th-systems-topic/cc-8th-systems-with-substitution/a/systems-of-equations-with-substitution.
13. Шкільняк С. С. Математична логіка. Приклади і задачі / Степан Степанович Шкільняк. – Київ: Видавничо-поліграфічний центр "Київський університет", 2007. – 145 с.

ДОДАТОК А

Таблиці класів, полів та методів

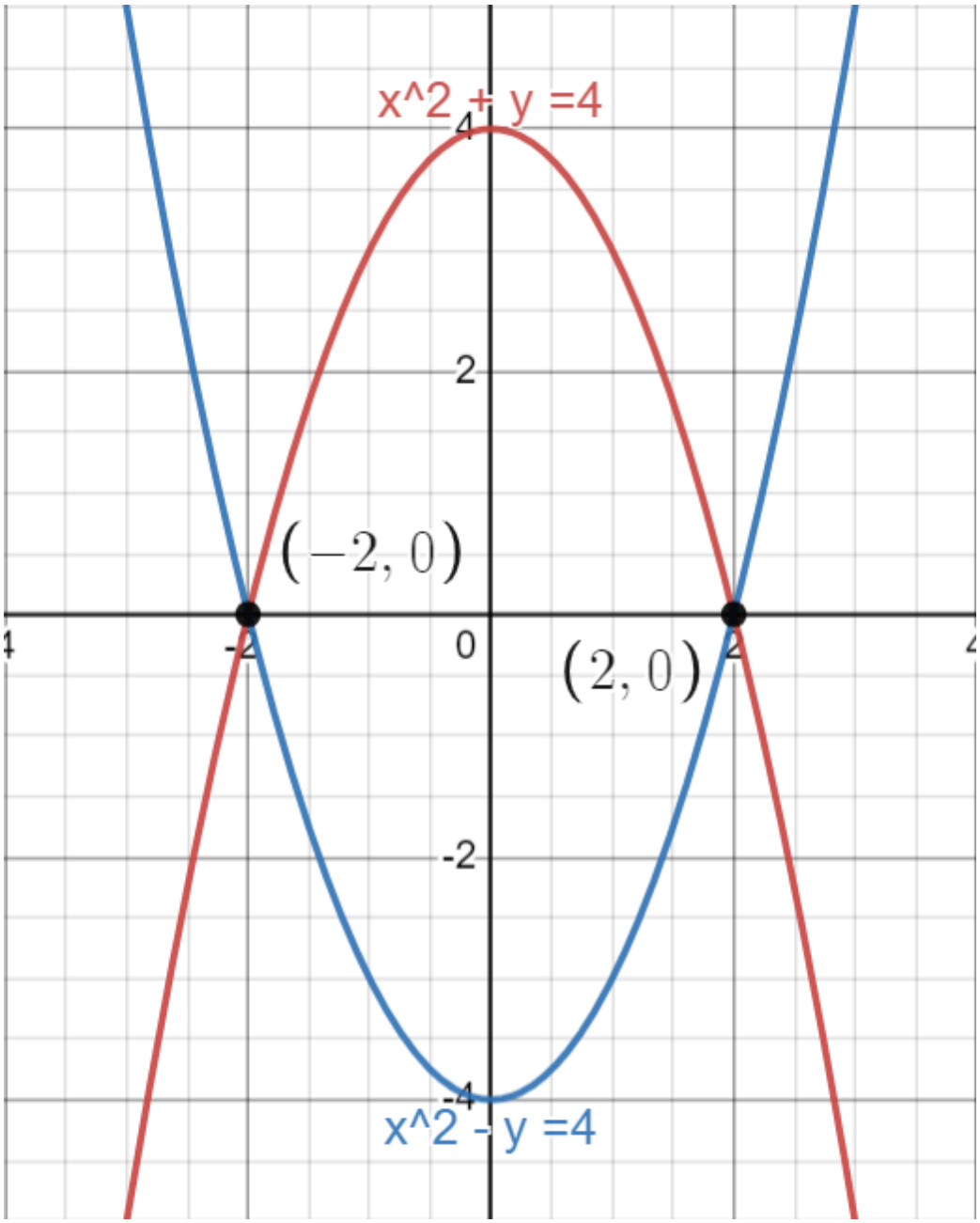



BooleanExpressionImply
BooleanExpressionImply(BooleanExpression, BooleanExpression)
inferRight(Sequent) List<Sequent>
equals(Object) boolean
hashCode() int
inferLeft(Sequent) List<Sequent>
BooleanExpressionAnd
BooleanExpressionAnd(BooleanExpression, BooleanExpression)
inferLeft(Sequent) List<Sequent>
inferRight(Sequent) List<Sequent>
BooleanExpressionOr
BooleanExpressionOr(BooleanExpression, BooleanExpression)
inferLeft(Sequent) List<Sequent>
inferRight(Sequent) List<Sequent>
BooleanExpressionGeometricStructure
BooleanExpressionGeometricStructure()
parse(String) BooleanExpressionGeometricStructure
inferLeft(Sequent) List<Sequent>
inferRight(Sequent) List<Sequent>
signs List<BooleanExpression>
properties List<BooleanExpression>
BooleanExpressionEquals
BooleanExpressionEquals(BooleanExpression, BooleanExpression)
inferLeft(Sequent) List<Sequent>
inferRight(Sequent) List<Sequent>
BooleanExpressionBinaryOperation
BooleanExpressionBinaryOperation(BooleanExpression, BooleanExpr
hashCode() int
equals(Object) boolean
toString() String
BooleanExpressionTriangle
BooleanExpressionTriangle(String, String, String)
cosineRule() List<Equality>
edgesAndAngles() List<Inequality>
sumOfAngles() List<Equality>
toString() String
hashCode() int
triangleInequality() List<Inequality>
equals(Object) boolean
signs List<BooleanExpression>
properties List<BooleanExpression>
BooleanExpressionNot
BooleanExpressionNot(BooleanExpression)
hashCode() int
toString() String
equals(Object) boolean
inferLeft(Sequent) List<Sequent>
inferRight(Sequent) List<Sequent>
BooleanExpression
BooleanExpression()
inferLeft(Sequent) List<Sequent>
toString() String
hashCode() int
skipBrackets(String, int) int
inferRight(Sequent) List<Sequent>
nextChar(String, int, Set<Character>) int
parse(String) BooleanExpression
clone() BooleanExpression
equals(Object) boolean
BooleanExpressionAtom
BooleanExpressionAtom(Solvable)
fact Solvable
toString() String
inferRight(Sequent) List<Sequent>
parse(String) BooleanExpressionAtom
inferLeft(Sequent) List<Sequent>
equals(Object) boolean
hashCode() int
fact Solvable


Sequent
Polynomial
Equality
SolutionsSet
SequentClosed
Solutions
Inequality
Solvable
Variable
SequentOpen
Constant
Solution
SolutionsNone
SolutionsAll

ДОДАТОК Б

Розв’язання системи рівнянь графічним методом



ДОДАТОК В

Проміжні обчислення програми

```
sequents size = 1
{ ⊢( x^2 + y ) = 4.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣y = 0.0 }
[{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣y = 0.0, ⊢( x^2 + y ) = 4.0
}]

sequents size = 1
{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣y = 0.0, ⊢( x^2 + y ) = 4.0 }
[{ ⊣y = 0.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0
}]

sequents size = 1
{ ⊣y = 0.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0 }
[{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0, ⊣y = 0.0
}]

sequents size = 1
{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0, ⊣y = 0.0 }
[{ ⊢( -4.0 + x^2 + ( ( ( x^2 * -1.0 ) + 4.0 ) * -1.0 ) ) = 0.0,
⊢( -4.0 + y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5
)^2 ) = 0.0, ⊢( -4.0 + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5
)^2 + ( y * -1.0 ) ) = 0.0, ⊢( ( x^2 * 2.0 ) + -8.0 ) = 0.0, ⊣y
= 0.0, ⊣( ( x^2 * -1.0 ) + 4.0 ) = 0.0, ⊣( x^2 + -4.0 ) = 0.0
}, { ⊢( -4.0 + x^2 + ( ( ( x^2 * -1.0 ) + 4.0 ) * -1.0 ) ) =
0.0, ⊢( -4.0 + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 )^2 + ( y
* -1.0 ) ) = 0.0, ⊢( -4.0 + y + ( sqrt(( ( ( y * -1.0 ) + -4.0
) * -4.0 )) * -0.5 )^2 ) = 0.0, ⊢( ( x^2 * 2.0 ) + -8.0 ) =
0.0, ⊣y = 0.0, ⊣( ( x^2 * -1.0 ) + 4.0 ) = 0.0, ⊣( x^2 + -4.0 )
= 0.0 }, { ⊢( -4.0 + x^2 + ( ( ( x^2 * -1.0 ) + 4.0 ) * -1.0 )
) = 0.0, ⊢( -4.0 + y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
```

)) * 0.5)^2) = 0.0, ⊢(-4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2 + (y * -1.0)) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊣y = 0.0, ⊣((x^2 * -1.0) + 4.0) = 0.0, ⊣(x^2 + -4.0) = 0.0 }, { ⊢(-4.0 + x^2 + (((x^2 * -1.0) + 4.0) * -1.0)) = 0.0, ⊢(-4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2 + (y * -1.0)) = 0.0, ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊣y = 0.0, ⊣((x^2 * -1.0) + 4.0) = 0.0, ⊣(x^2 + -4.0) = 0.0 }]

sequents size = 4

{ ⊢(-4.0 + x^2 + (((x^2 * -1.0) + 4.0) * -1.0)) = 0.0, ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢(-4.0 + (sqrt(((y + -4.0) * -4.0)) * 0.5)^2 + (y * -1.0)) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊣y = 0.0, ⊣((x^2 * -1.0) + 4.0) = 0.0, ⊣(x^2 + -4.0) = 0.0 } [{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

{ ⊢(-4.0 + x^2 + (((x^2 * -1.0) + 4.0) * -1.0)) = 0.0, ⊢(-4.0 + (sqrt(((y + -4.0) * -4.0)) * 0.5)^2 + (y * -1.0)) = 0.0, ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊣y = 0.0, ⊣((x^2 * -1.0) + 4.0) = 0.0, ⊣(x^2 + -4.0) = 0.0 } [{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

{ ⊢(-4.0 + x^2 + (((x^2 * -1.0) + 4.0) * -1.0)) = 0.0, ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢(-4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2 + (y * -1.0)) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊣y = 0.0, ⊣((x^2 * -1.0) + 4.0) = 0.0, ⊣(x^2 + -4.0) = 0.0 } [{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

```
{ ⊢( -4.0 + x^2 + ( ( ( x^2 * -1.0 ) + 4.0 ) * -1.0 ) ) = 0.0,
⊢( -4.0 + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 )^2 + ( y * -
1.0 ) ) = 0.0, ⊢( -4.0 + y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * -0.5 )^2 ) = 0.0, ⊢( ( x^2 * 2.0 ) + -8.0 ) = 0.0, ⊣y
= 0.0, ⊣( ( x^2 * -1.0 ) + 4.0 ) = 0.0, ⊣( x^2 + -4.0 ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

sequents size = 4
{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
[⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
[⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
[⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
[⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }]

sequents size = 4
⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
closed
⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
closed
⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
closed
⊤{ ⊢0.0 = 0.0, ⊣0.0 = 0.0, ⊣y = 0.0 }
Closed
```

ДОДАТОК Г

Проміжні обчислення програми

```
sequents size = 1
{ ⊢( x^2 + y ) = 4.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣( y + x )
= 0.0 }
[{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣( y + x ) = 0.0, ⊢( x^2 + y
) = 4.0 }]

sequents size = 1
{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊣( y + x ) = 0.0, ⊢( x^2 + y )
= 4.0 }
[{ ⊣( y + x ) = 0.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y
) = 4.0 }]

sequents size = 1
{ ⊣( y + x ) = 0.0, ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y )
= 4.0 }
[{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0, ⊣( y + x
) = 0.0 }]

sequents size = 1
{ ⊢( x^2 + ( y * -1.0 ) ) = 4.0, ⊢( x^2 + y ) = 4.0, ⊣( y + x )
= 0.0 }
[{ ⊢( ( y * -1.0 ) + -4.0 + ( sqrt(( ( y + -4.0 ) * -4.0 )) *
0.5 )^2 ) = 0.0, ⊢( -4.0 + y + ( sqrt(( ( ( y * -1.0 ) + -4.0 )
* -4.0 )) * 0.5 )^2 ) = 0.0, ⊢( ( x^2 * 2.0 ) + -8.0 ) = 0.0,
⊢( x^2 + ( ( ( x^2 * -1.0 ) + 4.0 ) * -1.0 ) + -4.0 ) = 0.0, ⊣(
y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0, ⊣( x^2 + -
4.0 + x ) = 0.0, ⊣( x + ( x^2 * -1.0 ) + 4.0 ) = 0.0, ⊣( y + (
sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }, {
⊢( ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 )^2 + -4.0
+ y ) = 0.0, ⊢( ( y * -1.0 ) + -4.0 + ( sqrt(( ( y + -4.0 ) * -
```

4.0)) * 0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }, { ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }, { ⊢((sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)^2 + -4.0 + y) = 0.0, ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

sequents size = 4
{ ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }, { ⊢0.0 =

0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }]

{ ⊢((sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)^2 + -4.0 + y) = 0.0, ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 } [{ ⊢0.0 = 0.0, ⊣2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }]

{ ⊢(-4.0 + y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)^2) = 0.0, ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 } [{ ⊢0.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣2.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }, { ⊢0.0 =

0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

{ ⊢((sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)^2 + -4.0 + y) = 0.0, ⊢((y * -1.0) + -4.0 + (sqrt(((y + -4.0) * -4.0)) * -0.5)^2) = 0.0, ⊢((x^2 * 2.0) + -8.0) = 0.0, ⊢(x^2 + (((x^2 * -1.0) + 4.0) * -1.0) + -4.0) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(x^2 + -4.0 + x) = 0.0, ⊣(x + (x^2 * -1.0) + 4.0) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣2.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0 }, { ⊢0.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

sequents size = 12
{ ⊢0.0 = 0.0, ⊣2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }

```
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) *
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y +
-4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -
4.0 ) * -4.0 )) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) *
-4.0 )) * 0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) *
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) *
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y
+ -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * -0.5 ) ) = 0.0 }]
```

```
{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣2.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y +
-4.0 ) * -4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * 0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣2.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * -0.5 ) ) = 0.0 }
```

```
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣2.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y
+ -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * -0.5 ) ) = 0.0 }]

sequents size = 12
{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 )
) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 )
) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -
4.0 ) * -4.0 )) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) *
-4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5
) ) = 0.0 }]
```

```
{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -
4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) *
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5
) ) = 0.0 }
```

```
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * 0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣-2.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) *
-4.0 )) * -0.5 ) ) = 0.0 }
[{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0
)) * -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -
0.5 ) ) = 0.0 }]

sequents size = 12
```

```
{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) =
0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) =
0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0, ⊣( y
+ ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 )
) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 )
) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* -0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5
) ) = 0.0 }
```

```
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0
}]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) =
0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) =
0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0 }
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊢0.0 = 0.0, ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 ))
* 0.5 ) ) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5
) ) = 0.0 }
[{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 )
) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) =
0.0 }]

{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
```

[⊥{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

{ ⊢0.0 = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }
[{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

sequents size = 12
{ ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }
[⊥{ ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0 }]

{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }
[⊥{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * 0.5)) = 0.0 }]

{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }
[⊥{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * 0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }]

{ ⊣(y + (sqrt((((y * -1.0) + -4.0) * -4.0)) * -0.5)) = 0.0, ⊣(y + (sqrt(((y + -4.0) * -4.0)) * -0.5)) = 0.0 }

```
[⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 )
) = 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) =
0.0 }]

sequents size = 4

⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
is not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
```

```
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
⊥{ ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0, ⊣( y
+ ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * 0.5 ) ) = 0.0 }
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * 0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
⊥{ ⊣( y + ( sqrt(( ( ( y * -1.0 ) + -4.0 ) * -4.0 )) * -0.5 ) )
= 0.0, ⊣( y + ( sqrt(( ( y + -4.0 ) * -4.0 )) * -0.5 ) ) = 0.0
}
not closed
```