ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДОВЕДЕННЯ ДЛЯ ЛОГІКИ ПРЕДИКАТІВ»

зі спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Виконавець роботи Кобець Антон Володимирович
_____ «___»_____2025 р.
(підпис)

Науковий керівник к. ф.- м. н., доцент Парфьонова Тетяна Олександрівна
_____ «___»_____2025 р.
(підпис)

Рецензент _____

ПОЛТАВА 2025

ЗМІСТ

ВСТУП

Сучасна комп’ютерна наука активно використовує формальні методи доведення тверджень, особливо в таких галузях, як штучний інтелект, формальна верифікація програмного забезпечення, обробка знань, математична логіка та моделювання інтелектуальних агентів. Однією з найважливіших логічних систем, що застосовуються для формалізації знань та виведення нової інформації, є логіка предикатів першого порядку. Доведення тверджень у логіці предикатів дозволяє отримувати нові знання з існуючих фактів і правил, що особливо актуально в умовах сучасної цифрової трансформації.

Із розвитком технологій виникає потреба в автоматизованих системах, які здатні працювати з формальною логікою та доводити або спростовувати твердження без втручання людини. Такі системи можуть бути застосовані для автоматизованого доказу теорем, створення експертних систем, автоматичного виведення нових знань із баз даних, формальної перевірки програмних специфікацій. Однак реалізація подібних систем потребує як чіткого математичного апарату, так і ефективної програмної реалізації.

У цій роботі реалізовано систему доведення тверджень у логіці предикатів, побудовану на основі методу резолюції, з урахуванням узгодження змінних та відстеження кроків виведення. Програма реалізована мовою Python з використанням графічної бібліотеки Tkinter для створення зручного та інтуїтивного інтерфейсу, що дозволяє користувачам формувати запити, вводити клаузи, переглядати хід доведення та результати обчислень. Для зручності поширення розроблене рішення було скомпільовано в автономний виконуваний файл із використанням PyInstaller.

Метою даної кваліфікаційної роботи є розробка ефективного програмного засобу для автоматизованого доведення тверджень у логіці предикатів, який включає в себе як основні алгоритми системи, так і зручний користувацький інтерфейс.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- вивчення існуючих підходів до реалізації доведення в логіці предикатів;
- аналіз алгоритмів уніфікації та резолюції;

- розробка архітектури системи та реалізація її ключових модулів мовою Python;
- створення графічного інтерфейсу користувача засобами Tkinter;
- реалізація відстеження кроків доведення;
- тестування роботи системи на практичних прикладах;
- створення виконуваного файлу для зручного розповсюдження програми.

Об’єктом дослідження є процес автоматизованого доведення логічних формул у логіці предикатів.

Предметом дослідження є програмна реалізація алгоритму резолюції з уніфікацією, засоби візуалізації процесу доведення та методи представлення логічної інформації у формалізованому вигляді.

Методи дослідження включають формальні методи логіки, методи алгоритмізації, структурного програмування та тестування програмного забезпечення.

Практичне значення роботи полягає у створенні наочного, простого у використанні інструменту, що дозволяє студентам, дослідникам та розробникам вивчати логіку предикатів, перевіряти логічні твердження для доведення у власних задачах. Розроблена система може бути застосована у навчальному процесі, в науково-дослідній діяльності та як основа для більш складних логічних чи експертних систем.

Наукова новизна роботи полягає в реалізації інтерактивного інтерфейсу для доведення логічних формул із можливістю відстежування всіх кроків резолюції, що підвищує прозорість та зручність аналізу логічного виведення. Система адаптована для освітніх цілей та легко масштабується під нові завдання.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

1.1. Актуальність задачі та теоритичне обґрунтування

Автоматизоване доведення тверджень у логіці предикатів першого порядку - це важлива задача комп’ютерної логіки, яка має широке застосування у сфері штучного інтелекту, логічного програмування, формальної верифікації програм, а також в системах підтримки прийняття рішень. У зв’язку зі зростанням складності інформаційних систем виникає потреба у створенні інструментів, які можуть автоматично, без втручання людини, перевіряти істинність висловлювань, знаходити суперечності у базах знань, або навіть генерувати нові логічні висновки на основі заданих фактів.

Існує безліч задач, які можуть бути представлені у вигляді логічних формул. Наприклад:

- перевірка достовірності певного твердження в математичній моделі;
- пошук логічних суперечностей у специфікації програмного забезпечення;
- виведення нових знань із формалізованих правил у базі знань експертної системи;
- доведення логічної коректності формул у рамках автоматичного доведення теорем.

Основна потреба у таких системах - це зниження людської участі у рутинних перевірках логічних структур, а також підвищення точності та швидкості логічного аналізу. Це особливо актуально у сферах, де помилки можуть мати критичні наслідки, наприклад у фінансових системах, у медичній діагностиці, або у розробці безпечного програмного забезпечення для авіації чи медицини.

Таким чином, задача автоматизації процесу доведення у логіці предикатів є важливою складовою побудови інтелектуальних комп’ютерних систем. Запропонована в цій роботі система розроблена з урахуванням вимог до гнучкості, прозорості обчислень, інтерактивності та доступності для кінцевого користувача.

Модель задачі, що реалізується у межах даної роботи, базується на формалізованому апараті логіки предикатів першого порядку, яка, у свою чергу, є природним розширенням класичної булевої логіки. На відміну від останньої, логіка предикатів дозволяє моделювати не лише істинні чи хибні твердження загального вигляду, але й конкретизувати зв'язки між об'єктами предметної області шляхом використання змінних, функціональних термів, предикатів та кванторів.

З формальної точки зору, модель включає такі ключові конструктивні елементи, характерні для логіки предикатів:

- терми (елементарні вирази у вигляді змінних, констант або функцій);
- літерали (атомарні логічні висловлювання або їх заперечення);
- клауза (диз'юнкція літералів у КНФ).

Всі ці елементи реалізовані як класи у Python: Term, Literal, Clause. Це дозволяє створити об'єктно-орієнтовану структуру, що сприяє модульності, повторному використанню та легкій підтримці коду.

Реалізовані також:

- уніфікатор змінних із перевіркою на рекурсивність (occurs-check);
- модуль інверсії запиту для приведення твердження до форми, придатної для доведення;
- механізм зберігання клауз та пошуку пар для резолюції;
- покрокове відстеження процесу доведення;
- обмеження кількості ітерацій для запобігання нескінченним циклам.

Узагальнено, ця частина дослідження формує методологічну базу для подальшої реалізації практичної системи.

1.2. Практична постановка задачі та характеристики системи

Метою є створення програмної системи автоматизованого доведення тверджень у логіці предикатів першого порядку, яка повинна бути зручною у використанні, мати прозору архітектуру та виводити цикл доведення - від простого введення тверджень до зрозумілого доведення або спростування.

Задача полягає в наступному:

1. Розробити внутрішню логічну модель, що реалізує механізми представлення та перетворення термів, літералів і клауз.
2. Реалізувати алгоритми уніфікації (включно з occurs-check) та резолюції для автоматичного логічного виводу.
3. Побудувати зручний графічний інтерфейс, у якому користувач може вводити клаузи та запити, запускати доведення та переглядати всі кроки виведення.
4. Реалізувати механізм покрокового журналу, який дозволяє візуально контролювати кожен етап логічного міркування.
5. Забезпечити ефективну архітектуру з модульним кодом, що дозволяє розширювати функціонал (наприклад, додати нові стратегії виводу або формати подання формул).
6. Підготувати систему до подальшої компіляції у виконуваний файл (.exe) для зручного поширення без залежностей.

Система буде реалізована на мові програмування Python. Для побудови графічного інтерфейсу обрано Tkinter - стандартну бібліотеку, яка дозволяє створювати прості у використанні GUI-додатки. Ключовими модулями програми є:

- класи для представлення логічних об’єктів (Term, Literal, Clause);
- функції для парсингу логічних формул із тексту;
- модуль уніфікації та резолюції з логічним контролем;
- система ведення журналу резолюційних кроків;
- інтерфейсна частина із можливістю взаємодії користувача з системою.

Очікуваний результат - інтерактивна програмна система, що дозволяє користувачу зручно вводити логічні твердження, аналізувати процес доведення, експериментувати з різними вхідними даними та отримувати вірні результати. Така система може бути корисною як у навчальних цілях, так і в прикладних задачах логічного аналізу.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Аналіз існуючих систем автоматизованого доведення

У галузі автоматизованого доведення тверджень існує низка відомих систем, які використовуються як у наукових дослідженнях, так і в комерційних або навчальних проєктах. Вони відрізняються підходами до реалізації логічного виводу, підтримкою формальних мов, інтеграцією з іншими інструментами та інтерфейсом користувача. Розглянемо кілька найпоширеніших.

Prolog – мова логічного програмування, яка включає механізм доведення тверджень на основі логіки предикатів. Prolog використовує уніфікацію та механізм зворотного висновку для пошуку рішень. Його переваги - зрозумілий синтаксис, активна спільнота користувачів, інтеграція з іншими мовами. Проте недоліком є складність масштабування на великі бази знань та обмеженість стандартних механізмів трасування.[6,10]

Vampire – це високоефективний автоматичний доводчик теорем, який підтримує логіку першого порядку з рівностями. Підходить для складних завдань, має підтримку TPTP-формату, використовує потужні стратегії пошуку. Його головні переваги - швидкодія, гнучкість конфігурації, підтримка багатьох стратегій доведення. Недоліки - складність налаштування та відсутність візуального інтерфейсу. [5,11,18]

Z3 – модульний доводчик від Microsoft Research, який підтримує перевірку тверджень у багатьох логіках, включаючи логіку предикатів. Має зручний API для Python, C++, Java. Використовується в автоматизованому аналізі програм, верифікації, розв’язанні обмежень. Переваги - надійність, активна підтримка, інтеграція з іншими інструментами. Недоліки - складність у використанні для новачків.[9,12]

Rocq – інтерактивне середовище для формального доведення теорем, засноване на інтуїціоністській логіці. Вимагає від користувача вручну формулювати та доводити твердження. Потужний інструмент для формальної верифікації, має велику екосистему. Недоліки - висока складність освоєння.[8,13]

Lean – сучасна система доведення, подібна до Rocq, але з акцентом на зручність та продуктивність. Активно використовується в математиці та формальних перевірках. Має дружній інтерфейс, мову Lean, активну спільноту. Підтримує інтерактивне доведення, автоматичне виведення та багато модулів. Запропонована система поєднує простоту введення та інтерфейсну доступність із прозорим відстеженням кроків доведення, що відрізняє її від більшості зазначених інструментів. Її основна перевага - навчальна орієнтація, прозорість, можливість візуального контролю за всіма етапами логічного виведення, що робить її ефективною у викладацькій практиці та самостійному вивченні логіки предикатів. [7,14]

У таблиці 2.1 наведено коротку характеристику цих систем.

Таблиця 2.1 – Огляд існуючих систем автоматизованого доведення

Назва	Тип логіки	Особливості	Переваги	Недоліки
Prolog	Логіка 1-го порядку	Мова програмування з уніфікацією	Простота, активна підтримка	Обмежене відстеження
Vampire	Автодоводчик	Підтримка TPTP, потужні стратегії	Швидкість, ефективність	Складне конфігурування
Z3	SMT-доводчик	API для різних мов, модульність	Інтеграція, стабільність	Високий поріг входу
Rocq	Конструктивна логіка	Інтерактивне доведення	Верифікація, потужність	Необхідність глибоких знань
Lean	Формальна логіка	Дружній інтерфейс, автоматизація	Простота використання, спільнота	Новизна, менша популярність

2.2. Аналіз алгоритмів автоматизованого доведення

Автоматизоване доведення у логіці предикатів базується на двох основних поняттях: уніфікації та резолюції. Саме ці механізми дозволяють здійснювати логічне виведення в рамках формальної системи.

Уніфікація – це процес, який знаходить підстановки для змінних у двох термах так, щоб зробити їх однаковими. Основою для цього є алгоритм уніфікації, що був вперше описаний у 1960-х роках Джоном Робінсоном. У нашій системі використовується рекурсивний алгоритм уніфікації з підтримкою перевірки на виникнення рекурсії, що запобігає нескінченним підстановкам.

Приклад уніфікації:

$P(X, a)$ і $P(b, a) \rightarrow X := b$

Резолюція - це правило виводу, що дозволяє із двох клауз вивести нову клаузу, яка логічно випливає з початкових. Резолюція працює з диз'юнкціями літералів і знаходить пари, які є протилежними за знаком, уніфікує їх, та створює нову клаузу без цих пар. У разі якщо результатом є порожня клауза - доведення завершено, твердження істинне.

Приклад:

1. $P(a) \vee Q(X)$
2. $\neg P(a)$

Застосовуючи правило резолюції до літералів $P(a)$ і $\neg P(a)$, отримуємо:

$\Rightarrow Q(X)$ - результат резолюції.

Переваги алгоритму резолюції:

- формальна коректність і повнота для логіки предикатів;
- простота реалізації;
- можливість механічного відстеження та контролю доведення;
- широке використання в системах логічного програмування.

Недоліки:

- потенційно велика кількість комбінацій пар для резолюції;

- ускладнення в оптимізації вибору пар для ефективного доведення;
- може не завершуватись в результаті відсутності рішення без додаткових обмежень на кількість кроків.

Для реалізації резолюції в нашій системі використовується алгоритм перебору пар клауз із динамічним доповненням нових клауз у загальну множину. Кожен крок фіксується у журналі доведення, що дозволяє детально проаналізувати хід доведення.

Таким чином, у розробленій системі реалізовано обидва алгоритми - уніфікацію та резолюцію - з урахуванням їх ключових властивостей, обмежень та ефективності для задач автоматичного доведення.

2.3. Технології для реалізації програм доведення

Для реалізації програмного забезпечення було обрано мову програмування Python, оскільки вона є однією з найбільш популярних і доступних мов, що забезпечують високу швидкість розробки, підтримку об’єктно-орієнтованого програмування, а також великий набір бібліотек для обробки тексту, графічного інтерфейсу та роботи з логічними структурами.

Одним з основних елементів інтерфейсної частини став Tkinter - стандартна бібліотека для створення графічного інтерфейсу в Python. Tkinter дозволяє швидко створювати віконні додатки з кнопками, полями введення, списками, мітками тощо. Завдяки своїй простоті та підтримці у всіх стандартних дистрибутивах Python, він є ідеальним вибором для створення навчального програмного забезпечення. Крім того, його можливості були достатніми для реалізації всіх функцій, передбачених у проєкті - введення формул, створення клауз, відображення історії запитів, перегляду результатів доведення.

Ще одним важливим інструментом став PyInstaller - засіб для конвертації Python-програм у виконувані файли. Це дозволило створити незалежний додаток, що не вимагає попередньо встановленого інтерпретатора Python на комп’ютері користувача. Це особливо зручно для поширення програмного продукту серед студентів, викладачів або науковців, які не мають спеціальних технічних навичок.

Також у проєкті використовувались:

- re - стандартний модуль Python для роботи з регулярними виразами. Застосовувався для парсингу логічних формул;
- collections.deque - структура даних з двобічною чергою, яка забезпечує ефективну реалізацію черги для перебору пар клауз під час резолюції;
- ScrolledText - компонент Tkinter, що дозволяє зручно відображати великі обсяги тексту для відстеження кроків доведення.

Таким чином, обрані технології дозволили реалізувати повноцінну програмну систему автоматизованого доведення, яка є зручною у використанні, легкою для розповсюдження, та ефективною для виконання поставлених задач. Її можна використовувати як у навчальному процесі, так і в якості інструменту для ознайомлення з принципами логіки предикатів.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Теоретичні основи логіки предикатів

Логіка предикатів першого порядку – це формальна система, яка дозволяє описувати висловлювання, що містять змінні, квантори, функції та предикати. Вона розширює висловлювальну логіку за рахунок введення структури предметної області, що дає змогу формалізувати складні твердження про об'єкти, їх властивості та відношення між ними.

Основні поняття логіки предикатів:

- Предикат – логічна функція, яка повертає істину або хибність залежно від значень аргументів. Наприклад, *P(x)* може означати «x - просте число».
- Терм – об’єкт, що може бути змінною (*X*, *Y*), сталою (*a*, *b*), або результатом функції (*f(X)*).
- Формула – предикат, до якого застосовано терми. Наприклад, *Likes(john, pizza)* – формула з двома аргументами.
- Літерал – найпростіший логічний вираз або його заперечення, наприклад ¬*P(x)*.

Квантори**:**

- Узагальнення (∀) - «для всіх». Наприклад, ∀*x*. *Mortal(x)* - усі x є смертними.
- Існування (∃) - «існує такий». Наприклад, ∃*x*. *Human(x)* - існує хоча б один x, що є людиною.

Формули логіки предикатів будуються за правилами граматики й можуть включати такі логічні зв’язки:

- ¬ (заперечення - не)
- ∧ (кон’юнкція - і)
- ∨ (диз’юнкція - або)
- → (імплікація)
- ↔ (еквівалентність)

Приклади формалізації висловлювань (табл. 3.1)

Таблиця 3.1 – Приклади формул, правила побудови та перетворення

Природна мова	**Формалізація**
Усі люди смертні	∀x (Human(x) → Mortal(x))
Існує студент, який вивчає математику	∃x (Student(x) ∧ Studies(x, Math))
Якщо воно літає, то це птах	∀x (Flies(x) → Bird(x))

Формули логіки предикатів будуються рекурсивно:

- терми комбінуються з предикатами для утворення атомарних формул;
- з атомарних формул за допомогою логічних зв'язків та кванторів утворюються складні формули.

Приклади побудови:

1. P(x) - атомарна формула: "x має властивість P".
2. ¬P(x) - заперечення: "x не має властивості P".
3. P(x) ∧ Q(x) - кон’юнкція: "x має властивості P і Q".
4. P(x) ∨ Q(x) - диз’юнкція: "x має принаймні одну з властивостей P або Q".
5. P(x) → Q(x) - імплікація: "якщо x має P, то має Q".
6. ∀x P(x) - універсальний квантор: "усі x мають властивість P".
7. ∃x P(x) - екзистенційний квантор: "існує принаймні один x, що має P".

Приклади перетворень:

1. Імплікація в диз’юнкцію: $P(x) \rightarrow Q(x) \equiv \neg P(x) \vee Q(x)$
2. Дистрибутивність: $P(x) \wedge (Q(x) \vee R(x)) \equiv (P(x) \wedge Q(x)) \vee (P(x) \wedge R(x))$
3. Закони де Моргана: $\neg(P(x) \wedge Q(x)) \equiv \neg P(x) \vee \neg Q(x)$ $\neg(P(x) \vee Q(x)) \equiv \neg P(x) \wedge \neg Q(x)$
4. Подвійне заперечення: $\neg(\neg P(x)) \equiv P(x)$

Приклади з джерела [2]: У лекціях з логіки предикатів наведено приклади комбінованих предикатів: $P(x) \wedge Q(x) \wedge R(x)$ - наприклад, "x – хлопець-студент, що навчається в університеті". Така формула відповідає перетину множин: $x \in P \cap Q \cap R$.

Крім класичної логіки предикатів, у практичному застосуванні логічних систем доведення використовуються скінченні автомати як формальні моделі. Вони дозволяють реалізовувати логічні функції у вигляді переходів між станами в залежності від вхідних предикатів. Це особливо актуально для автоматного перетворення вхідних логічних формул у вихідні твердження або реакції системи.

Згідно з формалізмом, предикати тут можуть розглядатися як оператори, що діють на послідовності символів вхідного алфавіту та формують відповідні вихідні логічні стани. Це дає змогу розглядати логіку предикатів не лише як абстрактну систему, а як частину логіки обчислювальних систем, що базуються на автоматах Мілі та Мура.

3.2. Алгоритм резолюції та уніфікації

Алгоритми резолюції та уніфікації є фундаментальними в автоматизованому доведенні логічних формул. Їх реалізація дозволяє знаходити логічні наслідки, здійснювати дедуктивне виведення та доводити або спростовувати твердження у рамках логіки предикатів першого порядку.

Уніфікація – це процес визначення такої підстановки значень змінних, яка дозволяє двом виразам мати однаковий вигляд. Наприклад, терми $P(X, a)$ і $P(b, a)$ можуть бути уніфіковані підстановкою $X := b$.

Алгоритм уніфікації включає:

1. Рекурсивну перевірку відповідності структури термів;

2. Застосування правил обробки змінних (перевірка на самозгадування - occurs-check);
3. Побудову множини підстановок.

Резолюція – це логічна операція, яка дозволяє з двох клауз, що містять протилежні літерали, створити нову клаузу. Цей процес продовжується доти, доки не буде отримано порожню клаузу (що означає істинність твердження) або вичерпано можливості.

Типовий крок резолюції:

Клауза 1: A ∨ B

Клауза 2: ¬B ∨ C

Результат: A ∨ C

У нашій системі реалізовано механізм автоматичного вибору пар клауз, що містять протилежні літерали, із подальшою уніфікацією їх аргументів та створенням нових клауз. Процес повторюється, і нові клаузи додаються до списку, доки не досягнуто результату.

Псевдокод алгоритму резолюції виглядає так:

```
while не досягнута порожня клауза:
    для всіх пар клауз (C1, C2):
        для кожної пари літералів (L1 ∈ C1, L2 ∈ C2):
            якщо L1 і ¬L2 уніфікуються:
                застосувати уніфікацію
                створити нову клаузу
                додати її до множини
    якщо не знайдено нових клауз:
        доведення неуспішне
```

Кожен крок у системі фіксується у журналі доведення, що дозволяє користувачеві відстежувати хід доведення. Це особливо корисно в навчальних цілях, коли важливо не лише отримати результат, а й розуміти логіку побудови доказу.

Використання цих алгоритмів у зв’язці дає змогу ефективно вирішувати задачі логічного виведення, формалізувати дедуктивний процес і забезпечити математично обґрунтовану реалізацію інтелектуального модуля доведення.

3.3. Блок-схема алгоритму системи доведення

Блок-схема (рисунок 3.1), яка лежить в основі програмної реалізації системи доведення, відображає загальну логіку взаємодії користувача з системою та внутрішні етапи обробки запиту. Кожен етап відповідає окремій функціональності, реалізованій у коді Python, та забезпечує послідовність виконання логічного доведення.

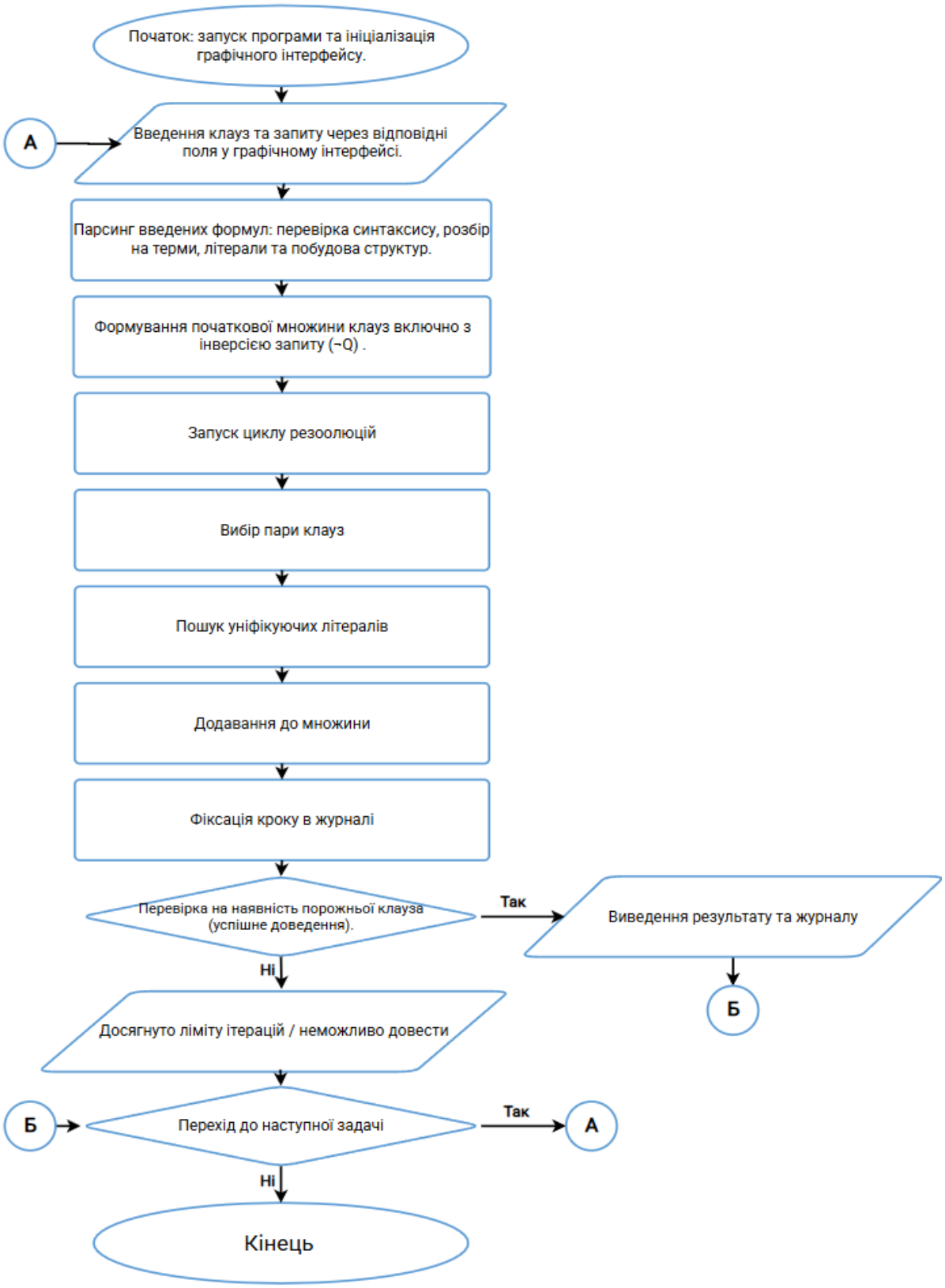

Рисунок 3.1 – Блок-схема роботи про програми

Текстовий опис блок-схеми:

1. Початок: запуск програми та ініціалізація графічного інтерфейсу.

2. Введення користувачем клауз та запиту через відповідні поля у графічному інтерфейсі.
3. Парсинг введених формул: перевірка синтаксису, розбір на терми, літерали та побудова структур.
4. Формування початкової множини клауз включно з інверсією запиту (¬Q) .
5. Запуск циклу резолюції:
    - Вибір пари клауз;
    - Пошук уніфікуючих літералів;
    - Побудова нової клаузи;
    - Додавання до множини;
    - Фіксація кроку у журналі.
6. Перевірка на наявність порожньої клаузи (успішне доведення).
7. Або досягнуто ліміту ітерацій / неможливо довести - завершення без доведення.
8. Виведення результату та журналу доведення в інтерфейсі.
9. Завершення роботи або повернення до нового введення.

Ця схема ілюструє структурну організацію системи, логіку переходів між етапами, та надає користувачу прозоре уявлення про те, як саме працює механізм доведення.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Інструменти та середовище розробки

Створення програмного забезпечення для автоматизованого доведення тверджень у логіці предикатів вимагає не лише вибору ефективного алгоритмічного підходу, а й обдуманого підбору інструментів реалізації, які забезпечують зручність, портативність та масштабованість системи. У даному розділі обґрунтовано вибір мови програмування, графічної бібліотеки для створення інтерфейсу та середовища розробки.

Вибір мови програмування – Python

Python було обрано як основну мову реалізації проєкту через такі причини:

- популярність і простота синтаксису, що дозволяє зосередитися на реалізації логіки, а не технічних деталях;
- підтримка об’єктно-орієнтованого програмування, що ідеально підходить для моделювання логічних конструкцій, таких як терми, літерали, клаузи;
- наявність великої кількості бібліотек, які спрощують обробку тексту, логіки, побудову інтерфейсу тощо;
- активна спільнота, що гарантує підтримку та наявність прикладів, документації, готових рішень.

Вибір бібліотеки для GUI – Tkinter

Tkinter – це стандартна бібліотека для створення графічних інтерфейсів у Python. Її вибір обґрунтовано наступним:

1. Стандартна поставка. Tkinter входить до стандартного дистрибутива Python, тому користувач не має потреби у додатковому встановленні - програма готова до запуску одразу після інсталяції Python.

2. Простота використання. Інтерфейс створюється за допомогою зрозумілого й мінімалістичного коду, що дозволяє зосередитися на логіці системи, а не на оформленні GUI.
3. Кросплатформеність. Додатки на Tkinter працюють на Windows, Linux, macOS без змін у коді.
4. Доступність документації та прикладів. Багата база знань та активна спільнота дає змогу швидко вирішувати проблеми, що виникають.
5. Повна інтеграція з Python. Це дозволяє легко об'єднати GUI з логічним модулем системи.

Вибір середовища розробки - Visual Studio Code

Для реалізації проєкту було обрано середовище Visual Studio Code (VS Code), яке має такі переваги:

1. Відкритість і безкоштовність. VS Code розповсюджується під ліцензією MIT і є абсолютно безкоштовним.
2. Система розширень. Існує велика кількість плагінів для Python, автоформатування, підсвітки синтаксису, запуску середовища тощо.
3. Кросплатформеність. VS Code однаково зручно працює на Windows, Linux і macOS.
4. Інтеграція з Git. Вбудовані інструменти для контролю версій спрощують командну роботу або створення історії змін.
5. Автоматичне доповнення та перевірка коду. Це підвищує швидкість та точність розробки.

Обрані інструменти – Python + Tkinter + Visual Studio Code – забезпечує оптимальний баланс між зручністю, гнучкістю, функціональністю та простотою в реалізації. Такий підхід дозволяє зосередитися на реалізації основної логіки доведення, не витрачаючи надмірних зусиль на технічні налаштування. Крім того, обрані інструменти є ідеальними для навчальних і демонстраційних цілей, що

важливо для подальшого застосування системи в освітньому процесі або самостійному вивченні логіки предикатів.

4.2. Розробка програмного забезпечення та графічного інтерфейсу

Програмне забезпечення реалізовано мовою Python із застосуванням структурованого підходу, що передбачає чіткий поділ на логічні модулі. Основною метою було забезпечити прозорість, модульність і зручність підтримки коду. Архітектура системи відокремлює логічне ядро (парсинг, уніфікація, резолюція) від частини інтерфейсу (GUI).

Загальна структура реалізації включає такі основні компоненти:

- Опис структур даних: реалізовано класи Term, Literal, Clause, які моделюють терми, літерали та диз'юнкції відповідно. Це дозволяє оперувати логічними об'єктами в зручній для програмування формі.
- Парсери: функції parse_term, parse_literal, parse_clause розбирають текстові формули та перетворюють їх у відповідні об'єкти логічної структури.
- Уніфікація: основні функції - unify, unify_var, occurs_check. Реалізація підтримує перевірку рекурсивного замикання (occurs-check), що підвищує коректність логічного виводу.
- Резолюція: функції resolve та resolution_traced реалізують алгоритм логічного виводу з трасуванням усіх кроків - це дозволяє користувачеві спостерігати процес побудови доведення.

Інтерфейс користувача побудовано за допомогою Tkinter, стандартної бібліотеки Python для створення графічних віконних застосунків. Вибір Tkinter обумовлено його простотою, відсутністю зовнішніх залежностей та кросплатформенністю. GUI було спроєктовано з урахуванням зручності для користувача, особливо - у навчальному контексті.

Ключові елементи інтерфейсу:

- Інструкція користувача: на початку вікна програми надані короткі правила синтаксису: великі літери - змінні, малі - константи та функції; логічне заперечення задається прапорцем.
- Редактор клауз: кожна клауза має окремий блок із назвою "Клауза №", де користувач вводить літерали. Кожен літерал супроводжується полем вводу, прапорцем для інверсії та кнопкою видалення.
- Кнопка "Додати нову клауза" - дає змогу додати ще одну диз’юнкцію літералів.
- Поле запиту: користувач вводить логічне твердження, істинність якого перевіряється.
- Кнопка "Перевірити": запускає процес доведення з відображенням усіх кроків у журналі.
- Журнал доведення: реалізований за допомогою компонента ScrolledText. Містить текстовий протокол усіх кроків резолюції з поясненнями підстановок та результатів.
- Історія запитів: дозволяє переглядати останні виконані доведення.

Такий підхід до організації інтерфейсу забезпечує інтуїтивну зрозумілість навіть для користувачів без попереднього досвіду роботи з логікою предикатів.

Нижче подано візуальні приклади ключових елементів інтерфейсу та приклад використання програми в різних сценаріях. Це дає змогу краще зрозуміти принципи роботи системи та взаємодії користувача з нею.

На рис 4.1 зображено головне вікно програми з кількома доданими клаузами. Чітко видно розмітку для літералів, кнопку «Перевірити», поле запиту та журнал.

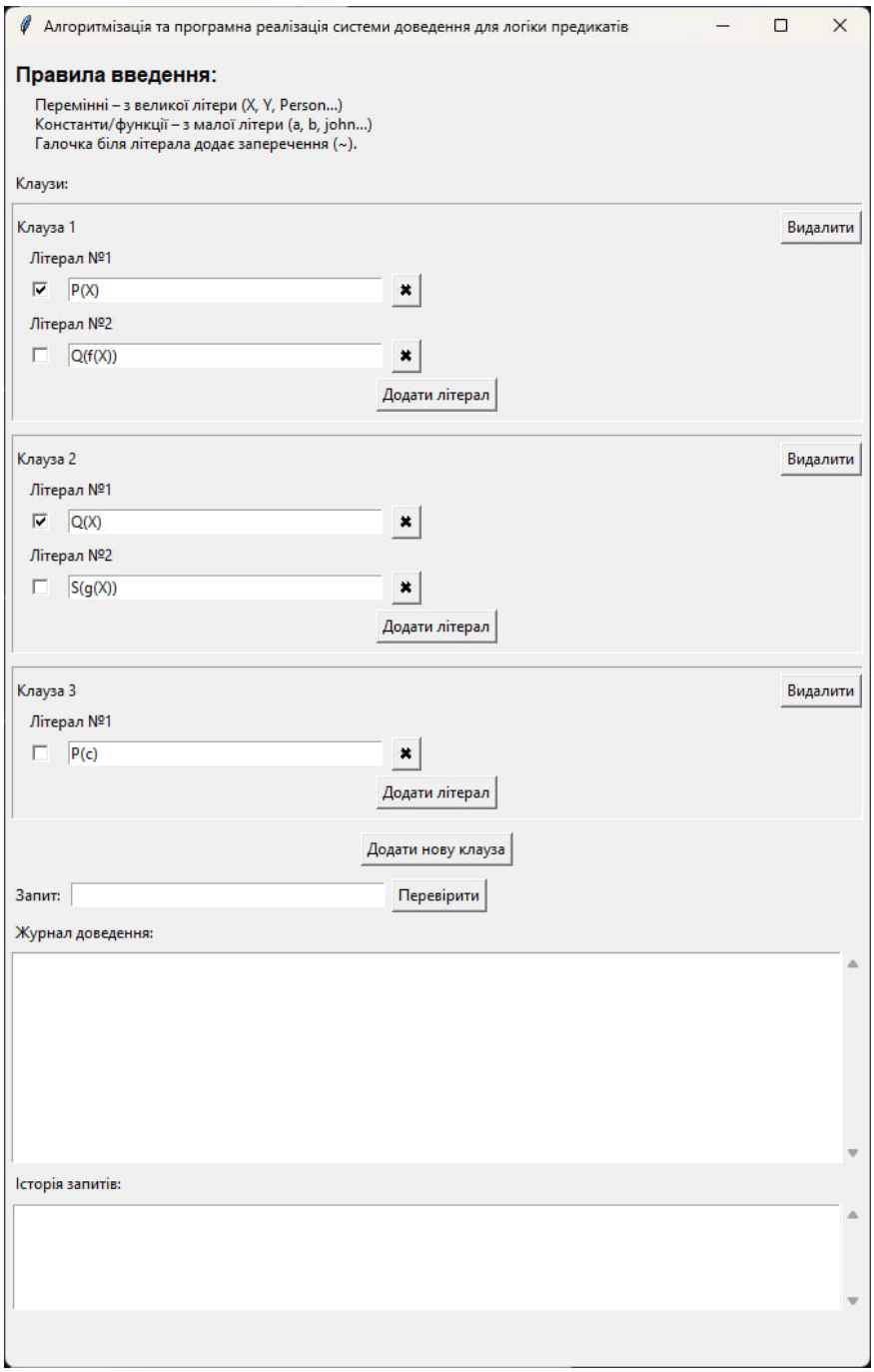

Рисунок 4.1 – Інтерфейс програми з даними

Логічне заперечення:

На рис 4.2 виділено, як користувач вказує заперечення – через галочку біля кожного літерала. Це дозволяє візуально швидко задавати диз’юнктивні заперечення.

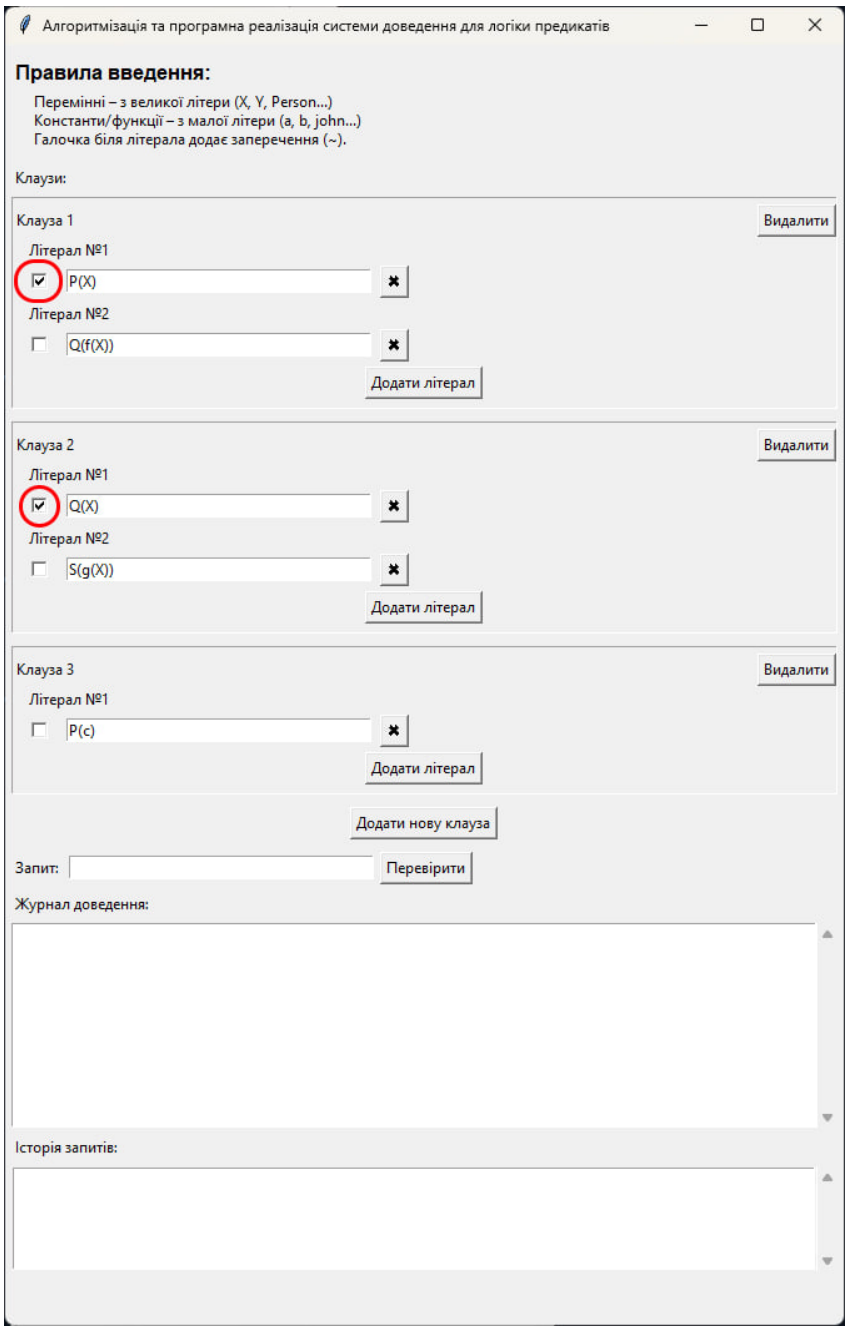

Рисунок 4.2 – Інтерфейс програми з виділеним функціоналом заперечення

Приклад результату доведення:

Рис. 4.3 демонструє результат натискання кнопки «Перевірити». У журналі показано:

- які клаузи об'єднувались,
- які підстановки були виконані,
- фінальний результат – у цьому випадку: Доведено ✔.

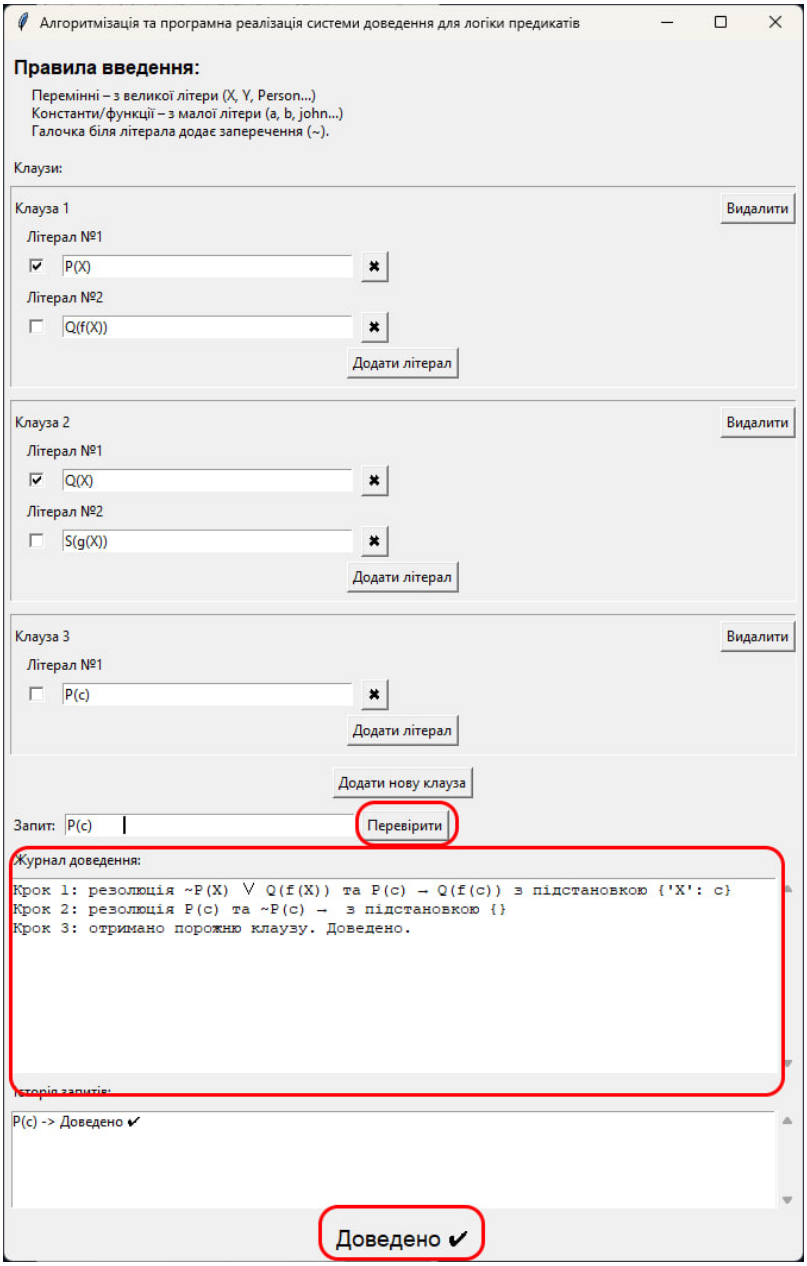

Рисунок 4.3 – Інтерфейс програми з виділеним функціоналом запуску, журналом доведення і результату

Завдяки такій реалізації інтерфейс є інтуїтивно зрозумілим навіть для користувачів без досвіду роботи з логікою предикатів або програмуванням.

4.3. Створення виконуваного файлу

Щоб зробити програму придатною до використання на будь-якому комп’ютері без необхідності встановлення Python та залежностей, було прийнято рішення зібрати виконуваний файл (.exe) за допомогою інструменту PyInstaller.

PyInstaller - це утиліта, яка дозволяє пакувати Python-програми у самодостатні виконувані файли. Такий файл містить інтерпретатор, усі модулі, ресурси, а також сам код, що дозволяє запускати програму навіть на машинах, де не встановлено Python.

Покроковий процес створення виконуваного файлу

1. Встановлення PyInstaller:

До командного рядку системі прописуємо – “pip install pyinstaller”. (рис. 4.4)

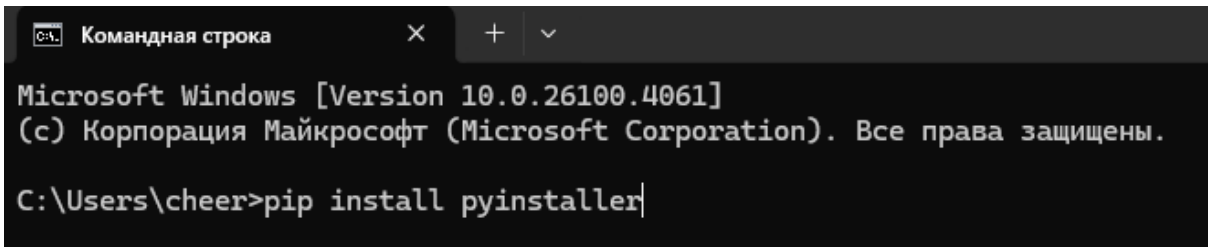

Рисунок 4.4 -Приклад введення запиту до командного рядку Windows

2. Підготовка скрипту: переконатися, що основний .py-файл містить усі імпорти та не вимагає додаткових ресурсів із зовнішніх директорій (усе має бути самодостатнім).
3. Команда для створення .exe: pyinstaller --onefile --windowed your_script.py. (рис. 4.5)

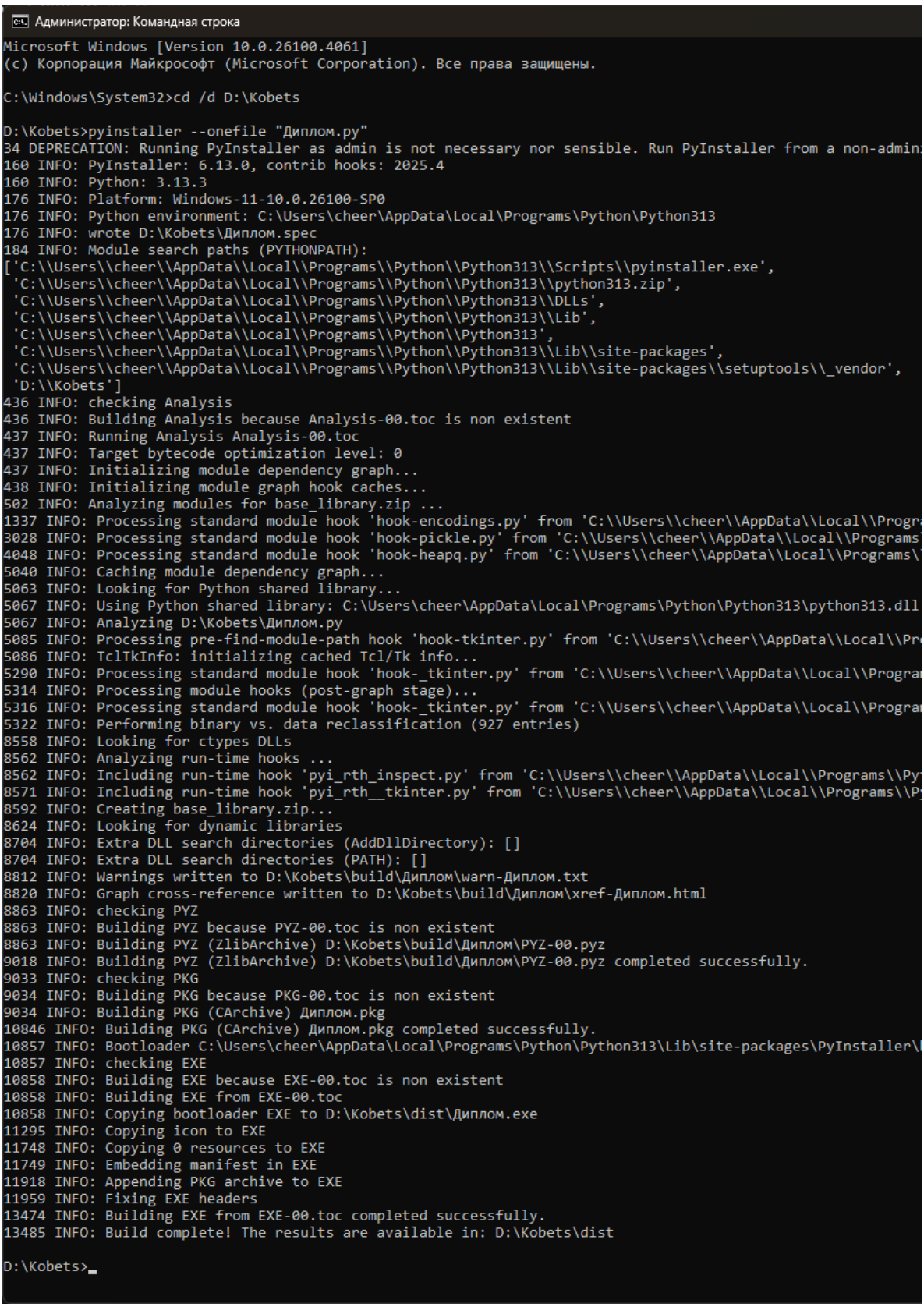

Рисунок 4.5 – Процес створення виконуваного файлу через командний рядок

- --onefile – створює один виконуваний файл (без додаткових папок);
- --windowed – приховує консоль (для GUI-додатків);

- your_script.py – назва основного Python-файлу.

4. Готовий файл з'являється у папці dist/ у каталозі проєкту. (рис. 4.6)

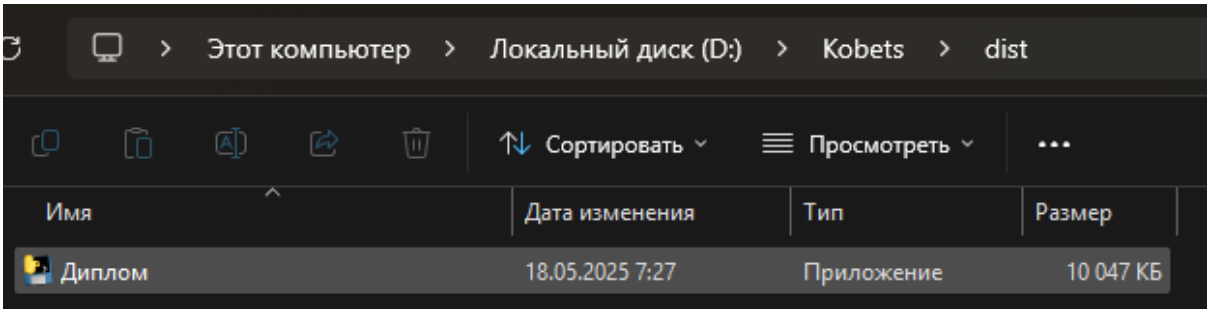

Рисунок 4.6 - Виконуваний файл у папці dist/ у каталозі проєкту dist/your_script.exe

5. Перевірка: виконуваний файл тестується на іншому комп’ютері, щоб перевірити, що він дійсно працює автономно.

Результатом є зручний графічний додаток, що відкривається подвійним кліком та не потребує встановлення Python або додаткових бібліотек. Це дозволяє легко поширювати розроблену систему серед викладачів, студентів або як навчальний інструмент у закритому середовищі.

4.4. Тестування: підхід, результати та приклади

Тестування є невід'ємною частиною розробки програмного забезпечення, особливо у випадку систем, що реалізують складні алгоритми логічного доведення. Метою тестування у рамках даної роботи було не лише перевірити технічну коректність реалізованих алгоритмів, а й забезпечити впевненість у стабільності, надійності та передбачуваності поведінки системи у різних умовах використання.

Під час створення системи автоматизованого доведення використовувались структурні тести, які охоплюють основні логічні сценарії. Основна увага приділялась перевірці резолюції, уніфікації, генерації порожньої клаузи, правильному обробленню заперечень, а також коректному формуванню журналу доведення. Особливу увагу також приділено взаємодії користувача з інтерфейсом - від формування клауз і запитів до інтерпретації отриманого результату. Простота і

наочність відображення проміжних кроків дозволила протестувати систему не лише на технічному рівні, але й з погляду зручності її використання.

Для тестування було підібрано ряд прикладів різної складності, що дозволило виявити переваги обраної архітектури та перевірити, як система поводиться при роботі з як простими, так і зі складнішими логічними формулами. Було підготовлено набір тестових прикладів, які охоплюють:

- прості предикати з однією змінною;
- предикати з вкладеними функціями;
- приклади з неоднозначною уніфікацією;
- ситуації, що призводять до порожньої клаузою

Приклад 1: простий предикат з однією змінною

- Клаузи: P(X), P(a)
- Запит: P(a)
- Журнал доведення. (рис. 4.7)

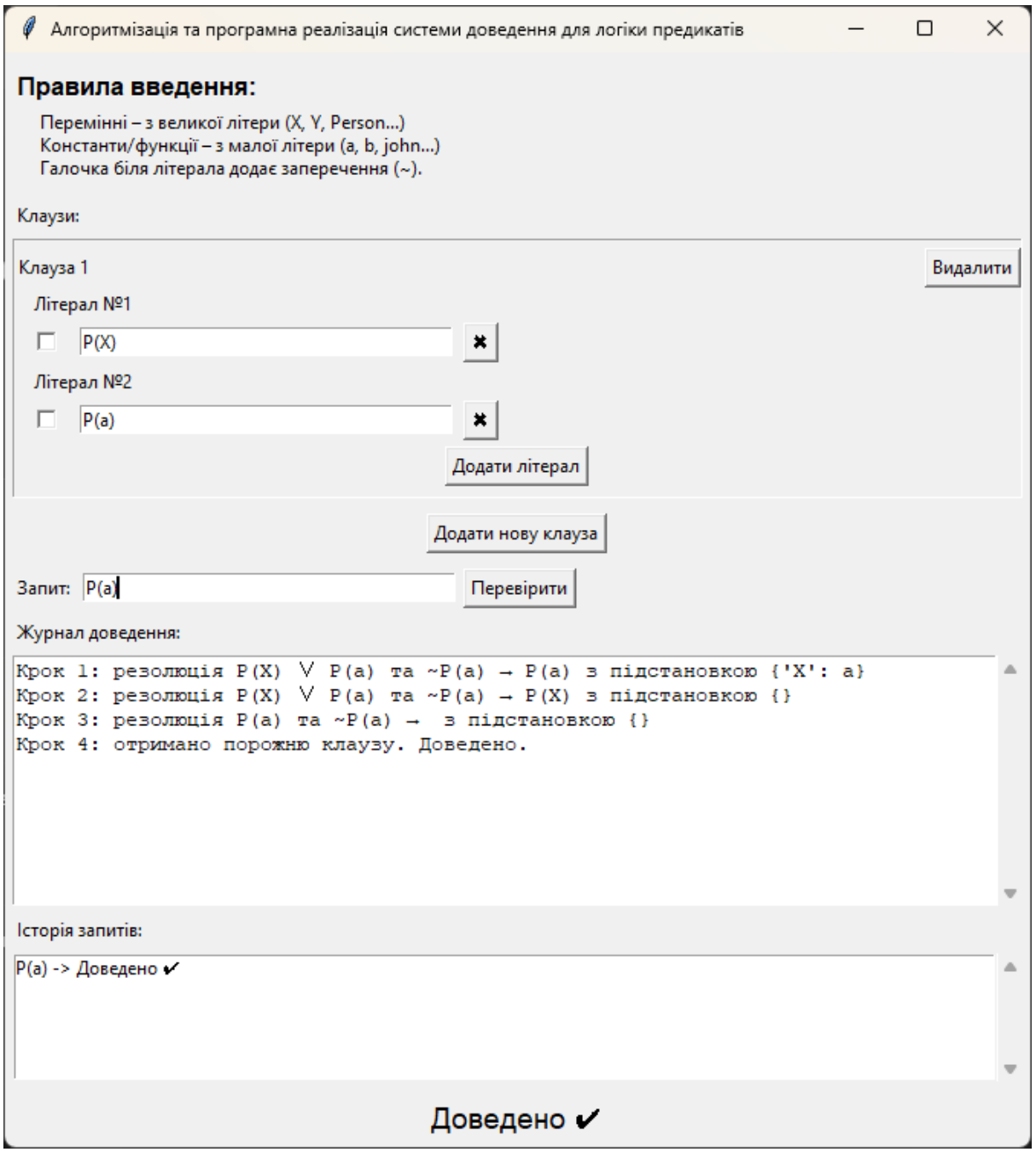

Рисунок 4.7 – Простий предикат з однією змінною

Приклад 2: предикат з вкладеною функцією

- Клауза: Q(f(g(X)))
- Запит: Q(f(g(a)))
- Журнал доведення: (рис. 4.8)

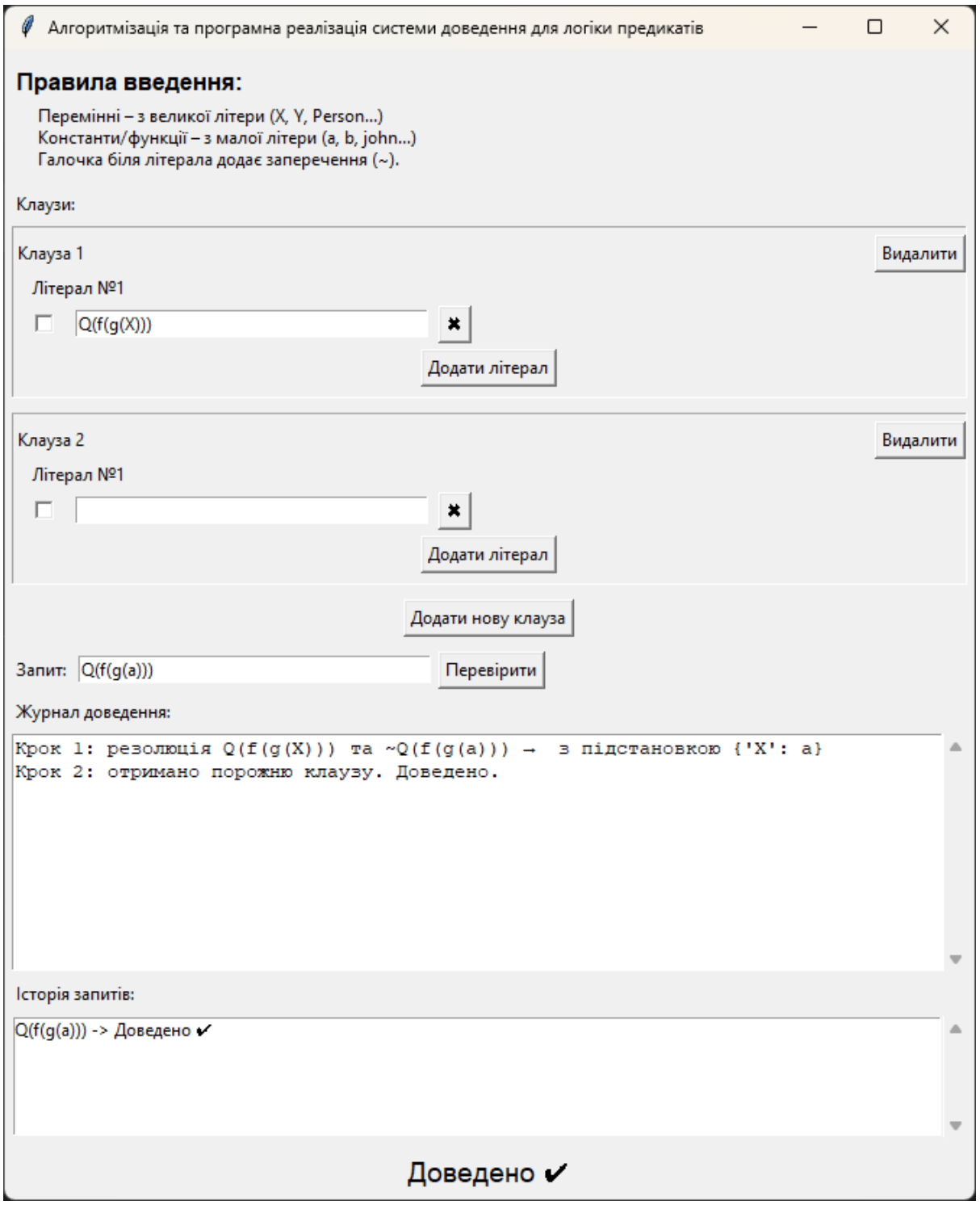

Рисунок 4.8 – Предикат з вкладеною функцією

Приклад 3: неоднозначна уніфікація

- Клаузи: R(X,a) ∨ R(b,Y)
- Запит: R(b,a)
- Журнал доведення: (рис. 4.9)

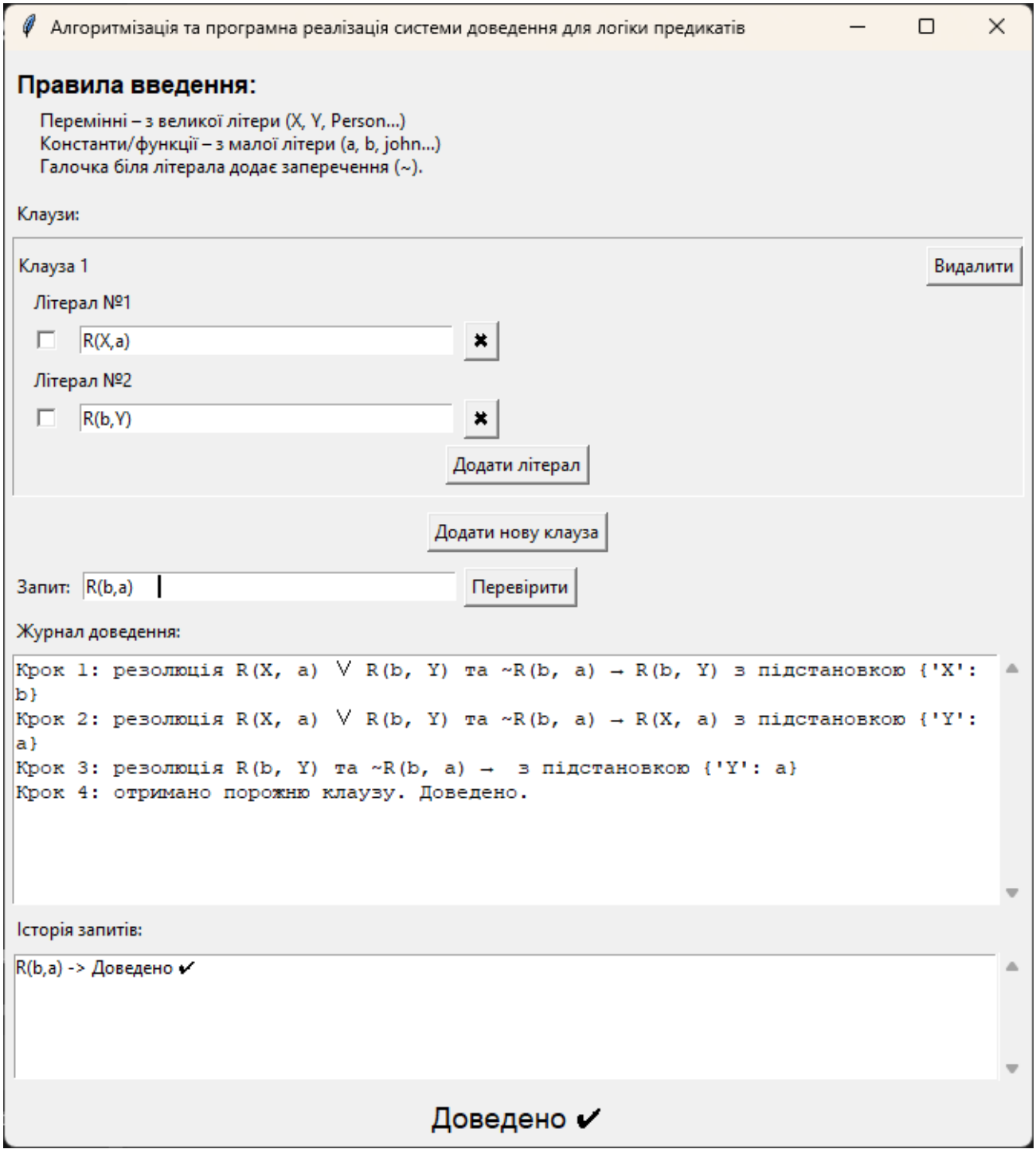

Рисунок 4.9 – Неоднозначна уніфікація

Приклад 4: порожня клауза

- Клаузи: P(a), ¬P(a)
- Запит: P(a)
- Журнал доведення: (рис. 4.10)

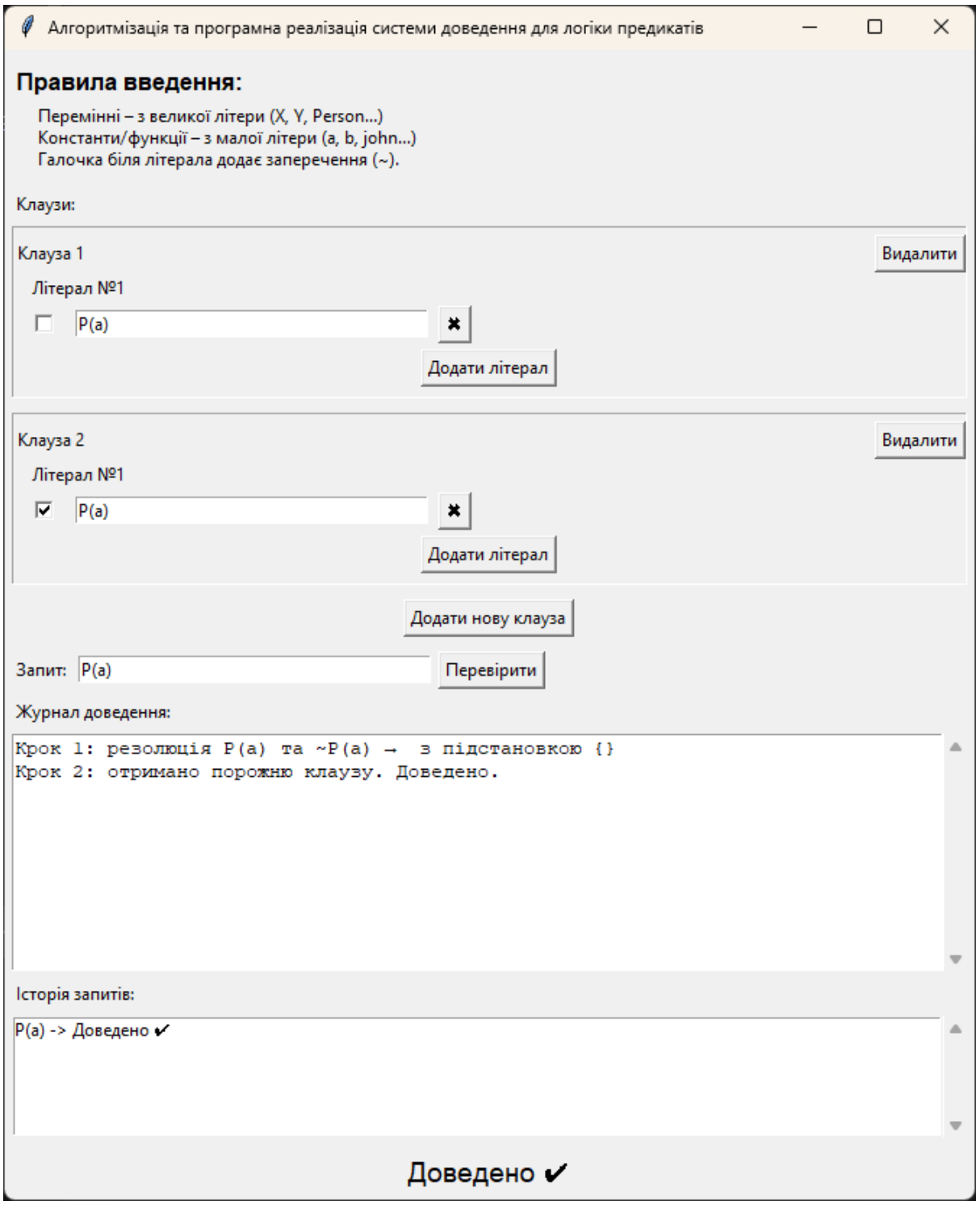

Рисунок 4.10 – Порожня клауза

Ці приклади підтверджують працездатність розробленої системи, коректність реалізації логіки резолюції та уніфікації, а також демонструють зручність інтерфейсу при роботі з доведеннями задач такого типу. Система продемонструвала стабільну та передбачувану поведінку навіть за умов збільшення обсягу вхідних даних. Основні

логічні операції - уніфікація термів, побудова нових клауз, застосування резолюції - виконуються з високою швидкістю. Всі приклади були оброблені миттєво, що вказує на ефективну реалізацію алгоритмів і належну оптимізацію структур даних. Таким чином, результати тестування підтверджують доцільність обраної архітектури, алгоритмічного підходу та практичну придатність системи для навчального і дослідницького використання.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано програму для доведення тверджень у логіці предикатів. Робота охоплює повний цикл - від постановки задачі до створення працездатного програмного інструменту. Реалізація поєднує елементи логічного виводу з базовим графічним інтерфейсом, що може бути використаний у навчальних цілях або для ознайомлення з принципами резолюції.

Особливу увагу приділено алгоритмам уніфікації та резолюції - фундаментальним методам автоматизованого логічного виводу, які в цій роботі реалізовані у чистому вигляді з усіма необхідними перевірками та супровідним відстеженням. Саме це дозволяє не тільки забезпечити точність доведення, а й зробити процес прозорим для користувача.

Розроблена система повністю функціональна, підтримує візуалізацію етапів доведення, що особливо важливо в навчальному контексті. Вона дозволяє не лише отримати відповідь на логічний запит, а й наочно прослідкувати шлях, яким ця відповідь була досягнута. Такий підхід сприяє формуванню в студентів аналітичного мислення, покращує розуміння логічних операцій, що є критично важливим у дисциплінах, пов’язаних з математичною логікою, штучним інтелектом та формальною інформатикою.

На практиці система показала стабільну та передбачувану роботу. Було проведено низку експериментів з різними конфігураціями вхідних даних, що підтвердило універсальність підходу та надійність обраної архітектури. Продуктивність при цьому залишалась на високому рівні навіть при обробці складних виразів, що свідчить про ефективну реалізацію основних обчислювальних алгоритмів. Крім того, практична перевірка функціональності довела відповідність програмного забезпечення очікуванням цільової аудиторії.

Крім технічного аспекту, важливою перевагою є зручний графічний інтерфейс, який робить взаємодію із системою доступною навіть для користувача без попереднього досвіду роботи з формальною логікою. Це відкриває широкі можливості використання програми в освітніх установах, на семінарах, лабораторних

роботах, а також як допоміжного інструменту для викладачів і науковців. Застосування системи у процесі навчання дозволяє зробити абстрактні теоретичні поняття більш наочними, а навчальний процес - більш інтерактивним і зрозумілим.

Загалом, результати дослідження підтверджують, що обрана архітектура, методологія реалізації, а також програмна платформа є доцільними для вирішення поставленої задачі. Створене програмне забезпечення володіє потенціалом до розширення, інтеграції в більші системи, а також використання у прикладних галузях, пов’язаних із логічним аналізом, штучним інтелектом, обробкою знань і формальною верифікацією.

У майбутньому, враховуючи модульну структуру програми, можливе доповнення новими функціональностями, зокрема:

- підтримкою логіки другого порядку;
- реалізацією альтернативних стратегій пошуку та доведення;
- інтеграцією з іншими логічними або експертними системами;
- розробкою веб-версії для онлайн-доступу;
- адаптацією інтерфейсу для мобільних пристроїв;
- впровадженням засобів логічної верифікації коду у розробницьких середовищах.

Отже, результати цієї кваліфікаційної роботи мають не лише навчальне та прикладне значення, а й можуть стати основою для подальших досліджень у галузі логічного програмування та автоматизованого виведення знань. Система відповідає сучасним вимогам до програмних засобів для логічного аналізу та може бути успішно використана як у академічному, так і в дослідницькому середовищі.

СПИСОК ЛІТЕРАТУРИ

1. Ольховська О. В.,Черненко О. О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 Комп`ютерні науки ступеня бакалавра: Полтава : ПУЕТ, 2025. – 58 с.
2. Ємець О. О., Парфьонова Т. О. *Дискретна математика: навчальний посібник*. - 3-тє вид., допов. і переробл. – Полтава: ПУЕТ, 2023. URL: http://dspace.puet.edu.ua/bitstream/123456789/12869/1/269-2046%20Емец%2c%20Парфьонова.pdf
3. Resolution for Predicate Logic. Technical University of Munich, 2023. URL: https://www.cs.cit.tum.de/fileadmin/w00cfj/tcs/2023ss/logic/Slides/lecture12-slides.pdf
4. Resolution for Predicate Logic. University of Western Ontario, 2017. URL: https://www.csd.uwo.ca/~mmorenom/cs2209_moreno/slide/lec16-predicate-resolution.pdf
5. First-Order Theorem Proving and Vampire. ResearchGate, 2024. URL: https://www.researchgate.net/publication/262359193_First-Order_Theorem_Proving_and_Vampire
6. Automated Theorem Proving for Prolog Verification. Université de La Réunion, 2024. URL: https://lim.univ-reunion.fr/staff/fred/Publications/24-MesnardMP.pdf
7. Canonical for Automated Theorem Proving in Lean. arXiv, 2025. URL: https://arxiv.org/pdf/2504.06239
8. Hammer for Coq – Automation for Dependent Type Theory. University of Warsaw, 2017. URL: https://www.mimuw.edu.pl/~lukaszcz/coqhammer.pdf
9. Z3 Theorem Prover. Вікіпедія, 2025. URL: https://uk.wikipedia.org/wiki/Z3_Theorem_Prover
10. SWI-Prolog – мова логічного програмування. SWI-Prolog, 2024. URL: https://www.swi-prolog.org/
11. Vampire – автоматичний доводчик теорем. GitHub, 2024. URL: https://github.com/vprover/vampire

12. Z3 – модульний доводчик від Microsoft Research. GitHub, 2024. URL: **https://github.com/z3prover/z3**
13. Rocq – інтерактивне середовище формального доведення. Rocq-Prover, 2024. URL: https://rocq-prover.org/
14. Lean – система доведення тверджень. Lean-lang, 2024. URL: https://lean-lang.org/
15. Правило резолюцій. Вікіпедія, 2025. URL: https://en.wikipedia.org/wiki/Resolution_(logic)
16. Логіка першого порядку. Вікіпедія, 2025. URL: https://uk.wikipedia.org/wiki/Логіка_першого_порядку
17. Unification Algorithm - Artificial Intelligence || Resolution. YouTube, 2025. URL: https://www.youtube.com/watch?v=MM8ck-9dcAQ
18. The Vampire Journey: Building a Theorem Prover for the 21st Century. YouTube, 2024. URL: https://www.youtube.com/watch?v=19tDmDv7fzE
19. Математична логіка та теорія алгоритмів. Навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2021. URL: https://ela.kpi.ua/bitstream/123456789/45670/1/Matemat_lohika.pdf
20. Логічне програмування. Навчальний посібник. Львів: Національний університет «Львівська політехніка», 2005. URL: https://pz.lpnu.ua/files/lisp/sheketa.pdf
21. Автоматизоване доведення теорем. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Автоматизоване_доведення_теорем

ДОДАТОК А. Повний код програми з коментарями

```
 import re
from collections import deque
# Ліміт ітерацій резолюції
MAX_RES_STEPS = 1000
# Спроба підключити Tkinter
try:
        import tkinter as tk
        from tkinter import messagebox
        from tkinter.scrolledtext import ScrolledText
        tk_imported = True
except ImportError:
tk_imported = False
# Структури: терм, літерал, клауза
class Term:
        def __init__(self, name, args=None):
                self.name = name
                self.args = args or []
                def is_var(self):
        return self.name[0].isupper() and not self.args
    def __repr__(self):
        return f"{self.name}({', '.join(map(str,self.args))})" if self.args else self.name
class Literal:
    def __init__(self, name, args, neg=False):
        self.name, self.args, self.neg = name, args, neg
    def __repr__(self):
        s = f"{self.name}({', '.join(map(str,self.args))})"
        return f"~{s}" if self.neg else s
class Clause:
    def __init__(self, literals):
        self.literals = literals
    def __repr__(self):
        return ' ∨ '.join(map(str,self.literals))
# Парсери
def parse_term(s):
    s = s.strip()
    m = re.match(r"(\w+)\((.*)\)", s)
    if m:
        name, args = m.group(1), m.group(2)
        parts, cur, depth = [], '', 0
        for ch in args + ',':
            if ch == '(': depth += 1; cur += ch
            elif ch == ')': depth -= 1; cur += ch
```

```
            elif ch == ',' and depth == 0: parts.append(parse_term(cur)); cur = ''
            else: cur += ch
        return Term(name, parts)
    return Term(s)
def parse_literal(s):
    s = s.strip()
    neg = s.startswith('~')
    if neg: s = s[1:]
    if '(' in s and s.endswith(')'):
        name, inside = s.split('(',1)
        inside = inside[:-1]
        args = [parse_term(x) for x in inside.split(',') if x.strip()]
    else:
        name, args = s, []
    return Literal(name, args, neg)
def parse_clause(s):
    parts = s.split('∨')
    return Clause([parse_literal(p) for p in parts if p.strip()])
# Уніфікація та резолюція з трасуванням
def unify(x, y, subs):
    if subs is None: return None
    if x == y: return subs
    if x.is_var(): return unify_var(x, y, subs)
    if y.is_var(): return unify_var(y, x, subs)
    if isinstance(x, Term) and isinstance(y, Term) and x.name==y.name and
len(x.args)==len(y.args):
        for a,b in zip(x.args,y.args):
            subs = unify(a,b,subs)
            if subs is None: return None
        return subs
    return None
def unify_var(var, x, subs):
    if var.name in subs: return unify(subs[var.name], x, subs)
    if x.is_var() and x.name in subs: return unify(var, subs[x.name], subs)
    if occurs_check(var, x, subs): return None
    new = subs.copy(); new[var.name] = x; return new
def occurs_check(var, x, subs):
    if x.is_var() and x.name==var.name: return True
    if isinstance(x, Term):
        return any(occurs_check(var,t,subs) for t in x.args)
    return False
def apply_subst(lit, subs):
    return Literal(lit.name, [substitute(a,subs) for a in lit.args], lit.neg)
def substitute(term, subs):
    if term.is_var() and term.name in subs: return subs[term.name]
```

```
    if term.args: return Term(term.name, [substitute(t,subs) for t in term.args])
    return term
def resolve(c1, c2):
    res = []
    for i, l1 in enumerate(c1.literals):
        for j, l2 in enumerate(c2.literals):
            if l1.name==l2.name and l1.neg!=l2.neg:
                subs = unify(Term(l1.name,l1.args), Term(l2.name,l2.args), {})
                if subs is not None:
                    new = [apply_subst(l,subs) for k,l in enumerate(c1.literals) if k!=i]
                    new += [apply_subst(l,subs) for k,l in enumerate(c2.literals) if k!=j]
                    uniq=[]
                    for lit in new:
                        if lit not in uniq: uniq.append(lit)
                    res.append((subs, Clause(uniq)))
    return res
def resolution_traced(clauses, query):
    trace = []
    trace_count = 0  # лічильник кроків для журналу
    negq = Clause([Literal(l.name, l.args, not l.neg) for l in query.literals])
    clauses_all = clauses + [negq]
    # черга всіх пар клауз
    queue = deque((clauses_all[i], clauses_all[j])
                  for i in range(len(clauses_all))
                  for j in range(i+1, len(clauses_all)))
    seen = set()
    steps = 0
    while queue and steps < MAX_RES_STEPS:
        c1, c2 = queue.popleft()
        steps += 1
        for subs, r in resolve(c1, c2):
            trace_count += 1
            trace.append(f"Крок {trace_count}: резолюція {c1} та {c2} → {r} з
підстановкою {subs}")
            if not r.literals:
                trace_count += 1
                trace.append(f"Крок {trace_count}: отримано порожню клаузу. Доведено.")
                return True, trace
            key = repr(r)
            if key not in seen:
                seen.add(key)
                clauses_all.append(r)
                for c in clauses_all:
                    queue.append((r, c))
    # якщо ліміт досягнуто
```

```
    if steps >= MAX_RES_STEPS:
        trace_count += 1
        trace.append(f"Крок {trace_count}: ліміт ітерацій ({MAX_RES_STEPS}) досягнуто. Складно довести.")
        return None, trace
    # якщо черга порожня - не доведено
    trace_count += 1
    trace.append(f"Крок {trace_count}: неможливо отримати порожню клаузу. Не доведено.")
    return False, trace
# Графічний інтерфейс
if tk_imported:
    history=[]
    def update_history():
        history_listbox.delete(0,tk.END)
        for q,res in history[-10:]: history_listbox.insert(tk.END,f"{q} -> {res}")    def run_prover():
        clauses=[]
        for frame,widget in clause_frames:
            lits=[]
            for neg_var,entry,*_ in frame['lits']:
                txt=entry.get().strip()
                if not txt: continue
                if neg_var.get(): txt='~'+txt
                lits.append(txt)
            if lits: clauses.append(parse_clause(' ∨ '.join(lits)))
        q=entry_query.get().strip()
        if not q:
            messagebox.showwarning('Увага','Введіть запит!'); return
        query=parse_clause(q)
        ok,trace = resolution_traced(clauses,query)
        # Виводимо журнал
        trace_text.config(state='normal')
        trace_text.delete('1.0', tk.END)
        for line in trace: trace_text.insert(tk.END, line+"\n")
        trace_text.config(state='disabled')
        if ok is None:
            result_var.set('Складно 🤔')
        else:
            result_var.set('Доведено ✔' if ok else 'Не доведено ✗')
        history.append((q,result_var.get())); update_history()
    root=tk.Tk()
    root.title('Алгоритмізація та програмна реалізація системи доведення для логіки предикатів')
```

```
    # Правила введення
    tk.Label(root, text='Правила введення:',
font=('Arial',12,'bold')).pack(anchor='w',padx=5,pady=(10,0))
    desc = """Перемінні – з великої літери (X, Y, Person...)
Константи/функції – з малої літери (a, b, john...)
Галочка біля літерала додає заперечення (~)."""
    tk.Label(root, text=desc, justify='left',
wraplength=500).pack(anchor='w',padx=20,pady=(0,10))
    # Клауза
    tk.Label(root, text='Клаузи:').pack(anchor='w',padx=5)
    clauses_container=tk.Frame(root); clauses_container.pack(fill='x')
    clause_frames=[]
    def add_clause():
        idx=len(clause_frames)+1
        cf=tk.Frame(clauses_container,bd=1,relief='groove',pady=5)
        cf.pack(fill='x',pady=5,padx=5)
        hdr=tk.Frame(cf); hdr.pack(fill='x')
        tk.Label(hdr,text=f'Клауза {idx}').pack(side='left')
        tk.Button(hdr,text='Видалити',command=lambda f=cf:
remove_clause(f)).pack(side='right')
        lt=tk.Frame(cf); lt.pack(fill='x',padx=10)
        frame={'lab':lt,'lits':[]}
        tk.Button(cf,text='Додати літерал',command=lambda f=frame:
add_literal(f)).pack(pady=2)
        clause_frames.append((frame,cf))
        add_literal(frame)
    def remove_clause(cf):
        for i,(f,w) in enumerate(clause_frames):
            if w==cf: w.destroy(); clause_frames.pop(i); break
        for j,(_,w) in enumerate(clause_frames,start=1):
w.winfo_children()[0].winfo_children()[0].config(text=f'Клауза {j}')
    def add_literal(frame):
        idx=len(frame['lits'])+1
        tk.Label(frame['lab'],text=f'Літерал №{idx}').pack(anchor='w')
       row=tk.Frame(frame['lab']); row.pack(anchor='w',pady=2)
        vv=tk.BooleanVar(); tk.Checkbutton(row,variable=vv).pack(side='left')
        e=tk.Entry(row,width=40); e.pack(side='left',padx=5)
        tk.Button(row,text='✖',command=lambda r=row,f=frame:
remove_literal(f,r)).pack(side='left',padx=2)
        frame['lits'].append((vv,e,row))
    def remove_literal(frame,row):
        for i,item in enumerate(frame['lits']):
            if item[2]==row: row.destroy(); frame['lits'].pop(i); break
        for k,(vv,e,r) in enumerate(frame['lits'],start=1): frame['lab'].winfo_children()[2*(k-
1)].config(text=f'Літерал №{k}')
```

```
    add_clause(); add_clause()
    tk.Button(root,text='Додати нову клауза',command=add_clause).pack(pady=5)
    # Запит
    frm=tk.Frame(root); frm.pack(fill='x',padx=5,pady=5)
    tk.Label(frm,text='Запит:').pack(side='left')
    entry_query=tk.Entry(frm,width=40); entry_query.pack(side='left',padx=5)
    tk.Button(frm,text='Перевірити',command=run_prover).pack(side='left')
    # Журнал доведення
tk.Label(root,text='Журнал доведення:').pack(anchor='w',padx=5)
trace_text=ScrolledText(root,height=10,state='disabled');
trace_text.pack(fill='both',padx=5,pady=5,expand=True)
    # Історія
    tk.Label(root,text='Історія запитів:').pack(anchor='w',padx=5)
    hf=tk.Frame(root); hf.pack(fill='both',padx=5,pady=5)
    sb=tk.Scrollbar(hf); sb.pack(side='right',fill='y')
    history_listbox=tk.Listbox(hf,height=5,yscrollcommand=sb.set)
    history_listbox.pack(side='left',fill='both',expand=True)
    sb.config(command=history_listbox.yview)
    result_var=tk.StringVar()
    tk.Label(root,textvariable=result_var,font=('Arial',14)).pack(pady=5)
    root.mainloop()
else:
print("Tkinter не встановлено.")
```

ДОДАТОК Б. Інструкція користувача

Запуск програми

1. Якщо ви використовуєте виконуваний файл (.exe), запустіть програму подвійним кліком по ярлику Диплом.exe.

2. Якщо ви запускаєте програму з Python-коду:

- переконайтесь, що у вас встановлений Python 3.7 або новіший;

- встановіть бібліотеки за потреби (наприклад tkinter, яка зазвичай є вбудованою);

- запустіть файл main.py через термінал або IDE (IDLE, PyCharm):

python main.py

Правила введення формул

- Змінні: позначаються з великої літери (X, Y, Person).

- Константи та функц -ії: з маленької літери (a, b, john, f(x)).

- Літерал вводиться у формі P(x,y) або ~P(x,y) (заперечення через тильду).

- Також логічне заперечення можна задати, встановивши галочку зліва від кожного літерала - це автоматично додасть символ ~ до предиката.

- Кожна клауза - диз’юнкція літералів: P(a) ∨ ~Q(X).

Інтерфейс програми

- Інструкція - у верхній частині вікна наведено правила введення та пояснення.

- Клаузи - кожна група літералів у окремому блоці. Додайте нові клаузи кнопкою "Додати нову клауза".

- Літерали - додаються у межах клаузи. Кожен має:

  - поле для введення (наприклад: P(a, X)),

  - прапорець для заперечення (додає ~ або знімає його),

  - кнопку для видалення літерала.

- Поле запиту - логічна формула, яку потрібно довести.

- Кнопка "Перевірити" - запускає процес доведення.

- Журнал доведення - текстова область для перегляду кроків резолюції.

- Історія запитів - перелік останніх результатів.

Покрокова інструкція користування

1. Запустіть програму.

2. У полі "Клаузи" натисніть "Додати нову клауза" для створення логічного твердження.

3. Введіть літерали:

- вкажіть ім’я предиката (наприклад: P(X));

- позначте галочкою, якщо потрібно заперечення - це додасть ~;

- додайте/видаліть літерали при потребі.

4. У полі "Запит" введіть твердження для перевірки.

5. Натисніть "Перевірити".

6. Перегляньте результати у журналі та історії запитів.

Поради та зауваження

- Введення обробляється у КНФ (кон'юнктивна нормальна форма).

- Програма автоматично будує інверсію запиту (¬Q).

- У випадку успішного доведення буде виведено "Доведено ✔".

- Якщо логічно не доведено - "Не доведено ✗".

- При перевищенні ліміту кроків - "Складно ".

- Уникайте помилок введення - неправильний синтаксис призведе до помилки парсингу.

Приклад використання

- Клаузи:

- P(X)

- ~P(a)

- Запит: P(a)

- Очікуваний результат: Доведено ✔

- Пояснення: система уніфікує X := a, отримує P(a) та ~P(a), після чого - порожня клауза.

Скріншот інтерфейсу (рис. Б.1)

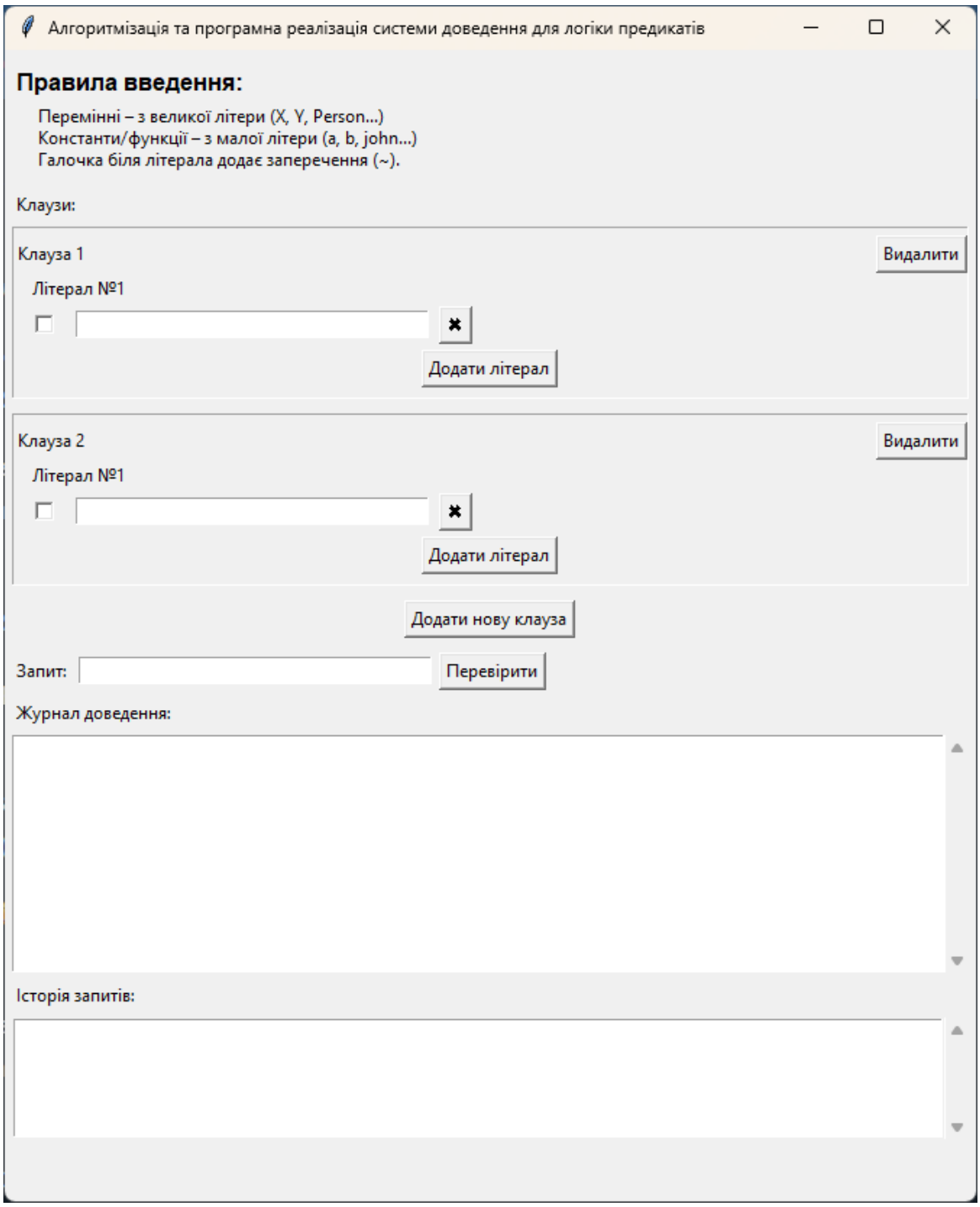

Рисунок Б.1 – Інтерфейс додатку

Програма орієнтована на навчальне використання та забезпечує прозору візуалізацію логічного доведення у логіці предикатів.

РЕФЕРАТ

Записка: 49 с., 12 рис., 2 таблиці, 2 додатки, джерела – 21.

Об’єкт дослідження – процес автоматизованого доведення логічних формул у логіці предикатів.

Мета роботи – розробка програмної системи для автоматизованого доведення тверджень у логіці предикатів з використанням методу резолюції та алгоритмів уніфікації, а також створення зручного графічного інтерфейсу для користувача.

Методи реалізації – програмна реалізація системи здійснена мовою Python із використанням бібліотеки Tkinter для побудови інтерфейсу користувача, PyInstaller для створення автономного виконуваного файлу, а також стандартних бібліотек re, collections і ScrolledText для обробки логічних структур, черг і відображення історії доведень. Реалізовано класи для логічних конструкцій (термів, літералів, клауз), уніфікації змінних та кроків резолюції.

У результаті тестування система показала правильність логічного виведення, зручність використання та можливість покрокового перегляду процесу доведення. Програма дає змогу користувачеві вводити логічні твердження у формалізованій формі, переглядати хід обчислень, аналізувати кроки резолюції та використовувати систему як інструмент навчання або дослідження.

Розробка цієї системи є актуальною у зв’язку зі зростанням потреби в автоматизованому логічному аналізі, особливо у сферах штучного інтелекту, верифікації програмного забезпечення, побудови експертних систем та підтримки прийняття рішень.

Записка: 53 с., 12 рис., 2 таблиці, 2 додатки, джерела – 21.

Об’єкт дослідження – процес автоматизованого доведення логічних формул у логіці предикатів.

Мета роботи – розробка програмної системи для автоматизованого доведення тверджень у логіці предикатів з використанням методу резолюції та алгоритмів уніфікації, а також створення зручного графічного інтерфейсу для користувача.

Методи реалізації – програмна реалізація системи здійснена мовою Python із використанням бібліотеки Tkinter для побудови інтерфейсу користувача, PyInstaller для створення автономного виконуваного файлу, а також стандартних бібліотек re, collections і ScrolledText для обробки логічних структур, черг і відображення історії

доведень. Реалізовано класи для логічних конструкцій (термів, літералів, клауз), уніфікації змінних та кроків резолюції.

У результаті тестування система показала правильність логічного виведення, зручність використання та можливість покрокового перегляду процесу доведення. Програма дає змогу користувачеві вводити логічні твердження у формалізованій формі, переглядати хід обчислень, аналізувати кроки резолюції та використовувати систему як інструмент навчання або дослідження.

Розробка цієї системи є актуальною у зв’язку зі зростанням потреби в автоматизованому логічному аналізі, особливо у сферах штучного інтелекту, верифікації програмного забезпечення, побудови експертних систем та підтримки прийняття рішень.