Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«Реалізація рекомендаційної системи для розв'язування математичних задач»**

Виконала: студентка 4-го року навчання,

Спеціальності

122 Комп'ютерні науки

Герасименко Єлизавета Олександрівна

Керівник Жежерун О.П.

кандидат фіз.-мат.наук, доцент

Рецензент_____

Кваліфікаційна робота захищена з оцінкою_____

Секретар ЕК_____

"_____" _____2025 р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

к.ф.-м.н., доц.

_____Жежерун О.П.

(підпис)

„____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студентки Герасименко Єлизавети Олександрівни факультету інформатики 4-го курсу

ТЕМА: **Реалізація рекомендаційної системи для розв’язування математичних задач**

Зміст ТЧ до кваліфікаційної роботи:

1. Індивідуальне завдання
2. Анотація
3. Вступ
4. Розділ 1. Теоретичні основи побудови рекомендаційних систем
5. Розділ 2. Аналіз предметної області та вибір онтологічної моделі
6. Розділ 3. Обробка природної української мови

7. Розділ 4. Теоретичні аспекти геометричних задач у онтологічних базах знань
8. Розділ 5. Реалізація алгоритму
9. Висновки
10. Список літератури
11. Додатки

Дата видачі “ ___” _____ 2025 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Реалізація рекомендаційної системи для розв’язування математичних задач

Календарний план виконання роботи:

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу.	15.10.2024	
2.	Огляд та аналіз технічної літератури за темою роботи.	20.01.2025	
3.	Визначення структури.	05.02.2025	
4.	Розробка алгоритму.	12.02.2025	
5.	Написання першого розділу.	22.02.2025	
6..	Написання другого розділу.	02.03.2025	
7.	Написання третього розділу.	15.03.2025	
8.	Написання четвертого розділу.	26.03.2025	
9..	Написання п’ятого розділу.	14.04.2025	

10.	Написання практичної частини.	01.04.2025	
11.	Написання висновків.	23.05.2025	
12.	Створення презентації та написання доповіді.	25.05.2025	
13.	Корегування роботи.	26.05.2025	
14.	Здача кваліфікаційної роботи.	01.06.2025	
15.	Захист кваліфікаційної роботи.	05.06.2025	

Студент Герасименко Є.О.

Керівник Жежерун О.П.

“_____”_____

Зміст

Анотація

У кваліфікаційній роботі розглянуто побудову рекомендаційної системи для розв’язування геометричних задач, що базується на онтологічному підході та обробці природної української мови. Основною метою дослідження є створення інтелектуального інструменту, який допомагає учням та викладачам ефективно орієнтуватися в тематиці задач з геометрії, зокрема теми “Трикутник", на основі глибокого розуміння змісту задачі.

У першому розділі описано теоретичні засади рекомендаційних систем, зокрема їхнє визначення, сфери застосування в освіті та класифікацію за принципами функціонування.

Другий розділ присвячено аналізу предметної області - геометричних задач, особливостям їх структури, а також вибору відповідної онтологічної моделі. Подано основні поняття баз знань, онтологій, розглянуто структуру онтології та інструменти для її створення.

Третій розділ присвячено проблемам обробки природної української мови, які виникають при автоматичному аналізі математичних задач. Описано сучасні методи NLP та обґрунтовано вибір інструментів для морфосинтаксичного аналізу, таких як UDPipe.

У четвертому розділі проаналізовано особливості формалізації геометричних задач в онтологічному вигляді, здійснено типологію задач, а також наведено приклад побудови онтологічної моделі конкретної задачі.

П’ятий розділ описує конкретні аспекти розробки досліджуваної системи, описує архітектуру класів, взаємодію з онтологією та мовними моделями, методику уніфікації логіки та підходи до обробки вхідних та вихідних даних.

Результатом дослідження є концептуальна модель та практична реалізація

прототипу рекомендаційної системи, що демонструє можливість використання онтологічних підходів і технологій обробки природної мови для автоматизованої підтримки навчального процесу у галузі шкільної геометрії.

Вступ

Геометрія - це один з ключових підрозділів математики, який звісно є невід'ємною складовою шкільної програми для учнів. Вона включає в себе не просто сухе вивчення теорем та правил, а й формування просторового та логічного мислення, уявлення про навколишній світ у поняттях фігур та предметів, їх розміри і взаємодію між собою. Більше того, знання з геометрії використовується усюди: в архітектурі, інженерії, спорті, навігації, вивченні природних закономірностей, тощо.

Хоча у цього предмету багато застосувань у реальному житті, через його багатогранність та складність, він є одним з найважчих у програмі. Багато учнів часто стикаються з недостатнім розуміння матеріалу і проблемою розвинення абстрактного мислення. В той же час, існуючі методи навчання геометрії не є ефективними - вони скоріше ускладнюють її вивчення.

Саме цим і зумовлена потреба в альтернативному підході з використанням сучасних технологій. Існування інструменту, який зробив би цей шлях більш цікавим та захоплюючим, який би мотивував учнів не просто знайти відповідь до задачі і забути, а власне зрозуміти і в майбутньому мати змогу власноруч розв'язувати поставлені завдання, став би революцією в автоматизації навчального процесу. Одним із таких інструментів є рекомендаційна навчальна система.

Рекомендаційні навчальні системи - це програми або алгоритми, які використовують теорію з предмету, приклади рішення задач та/або штучний інтелект для валідації відповідей учнів і рекомендацій щодо вирішення завдань. Використання такої системи для вивчення геометрії створить таке інтерактивне середовище навчання, яке стало б тим вирішенням проблеми, та покращило би навички розв'язування геометричних задач учнями без залучення допомоги батьків та репетиторів. Саме тому створення такої системи для розв'язку

математичних задач стала основною ідеєю та темою даної роботи.

Мета роботи: дослідити методи та особливості управління онтологіями та онтологічними базами знань у контексті створення рекомендаційної системи для розв’язку математичних задач. Конкретними завданнями для цього є вивчення і аналіз вигляду геометричних задач, теоретичних основ для їх вирішення, побудову моделей геометричних об'єктів в онтологічних базах знань, а також розробку та впровадження алгоритмів для розв’язання практичних задач із використанням цих баз.

Мета практичної частини: використання редактору онтологій з відкритим кодом, фреймворку для побудови інтелектуальних систем Protege для розробки рекомендаційної системи для розв’язування математичних задач.

Кінцевим результатом стане аналіз та підведення підсумків щодо ефективності використання рекомендаційної системи на основі онтологічної бази знань як додаткового ресурсу для навчання.

Предметом дослідження є процес формалізації та автоматичного розв’язування геометричних задач з використанням семантичного аналізу та онтологічних баз знань.

Об’єктом дослідження це онтологічна система, що описує зв’язки між геометричними об’єктами, їх властивості та правила та дозволяє отримувати судження з геометричних задач.

Актуальність роботи зумовлена активним розвитком інтелектуальних освітніх систем та штучного інтелекту. У сучасному світі активно зростає необхідність в можливостях автоматичної формалізації та обробки текстових знань. Розробки системи для пошуку та пояснення алгоритму розв’язку простих геометричних задач, важлива як для впровадження нових методів в навчальні процеси так і для популяризації можливостей семантичного, морфологічного

аналізу й роботи з онтологічними підходами обробки даних.

Наукова новизна роботи ґрунтується на можливостях динамічно обробляти текстові дані й формувати результати на їх основі, при цьому підтримуючи багаторазове розширення. На відміну від класичних систем комп'ютерної алгебри запропонований підхід рекомендаційної системи надає семантичну інтерпретацію об’єктів задачі поєднуючи це з онтологічним підходом для формулювання етапів розв’язку. Створена архітектура базується на морфологічному аналізі тексту та його лематизації та співставленні його з онтологічним представленням даних.

Розділ 1. Теоретичні основи побудови рекомендаційних систем

1.1 Поняття рекомендаційної системи та її застосування в освіті

Рекомендаційна система - це програмна система, яка оперує на базі штучного інтелекту та має на меті передбачити та надати нову інформацію на базі існуючої [9]. Такі системи допомагають користувачам знайти саме ті ресурси або рішення, які найбільше відповідають їхнім потребам, інтересам чи цілям.

Поява рекомендаційних систем стала відповіддю на виклики, пов’язані з перенасиченням інформаційного простору. З розвитком інтернету обсяги доступних даних зростали експоненційно, ускладнюючи процес пошуку релевантної інформації. Наприклад, на комерційних платформах, таких як Amazon або eBay, значне розширення асортименту товарів призвело до необхідності оптимізувати користувацький досвід. Просте сортування за категоріями стало недостатнім - виникла потреба у системах, які здатні аналізувати поведінку покупця і передбачати, які саме товари можуть його зацікавити. Це дозволило зменшити час пошуку, підвищити ефективність вибору та, відповідно, збільшити прибуток компаній [9].

Згодом алгоритми персоналізованих рекомендацій вийшли за межі електронної комерції. Сьогодні вони застосовуються в стримінгових сервісах (Netflix, YouTube), соціальних мережах (Facebook, TikTok), новинних агрегаторах (Google News), музичних платформах (Spotify) та мобільних додатках. Завдяки обробці великих обсягів даних, машинному навчанню та методам обчислювального інтелекту, сучасні системи можуть не лише відображати уподобання користувача, а й адаптуватися до його змін у режимі реального часу [9].

Однією з перспективних сфер застосування рекомендаційних систем є освіта. В умовах цифрової трансформації навчального процесу зростає потреба в

інструментах, які дозволяють індивідуалізувати підхід до кожного учня.

Освітні рекомендаційні системи здатні:

- підказувати найефективніші стратегії навчання на основі стилю мислення учня;
- підібрати завдання відповідного рівня складності;
- пропонувати навчальні матеріали, які сприяють глибшому засвоєнню знань;
- аналізувати помилки та допомагати уникнути їх у майбутньому [1].

У контексті вивчення геометрії такі системи особливо корисні. Геометричні задачі часто потребують не лише знання теорії, а й уміння застосовувати просторове мислення, логіку, абстрагування. Система, яка може аналізувати, на якому етапі учень застряг, і надати відповідні підказки чи рекомендовані теореми або методи розв’язання, значно підвищить ефективність навчання [1].

Метою даної роботи є створення рекомендаційної системи, що працює на основі онтологічної бази знань та допомагає школярам у процесі розв’язання геометричних задач. Така система має на меті не просто автоматизувати відповіді, а забезпечити пояснення, рекомендації та напрямки мислення, адаптовані під рівень знань користувача.

1.2 Види рекомендаційних систем

Рекомендаційні системи можна класифікувати за типом алгоритмів, які вони використовують. Найпоширенішими є такі підходи, як колаборативна фільтрація, контентно-орієнтовані системи, гібридні системи, та системи на основі знань.

1. Колаборативна фільтрація (Collaborative Filtering):

Цей підхід ґрунтується на припущенні, що користувачам, які раніше мали подібні вподобання, можуть подобатися схожі елементи в майбутньому.

Наприклад, якщо двоє учнів добре впоралися з однаковими геометричними задачами, то система може запропонувати одному з них задачу, яку вже розв’язав інший. Перевагою такого виду є відсутність потреби в знаннях про самі об'єкти, лише про взаємодію користувачів з ними, хоча при недостатній кількості даних фільтрація працює слабко [9].

2. Контентно-орієнтовані системи (Content-Based Filtering):

У таких системах аналізуються характеристики самого контенту та вподобання користувача. Наприклад, якщо учень часто розв'язує задачі на вписані кути, система пропонує йому подібні задачі, які мають ті самі властивості. Ці системи ефективно працюють навіть для нових користувачів, проте система може "замкнутися" на одних і тих самих типах задач [9].

3. Гібридні системи (Hybrid Systems):

Вони поєднують переваги попередніх двох методів, дозволяючи компенсувати їх недоліки. Наприклад, спочатку система аналізує тематику задач (контент), а далі порівнює результати учня з іншими користувачами (колаборація). Це найпоширеніший підхід у сучасних системах [9].

4. На основі знань (Knowledge-based systems):

Використовують експертні правила, логіку та структури знань. Такий підхід особливо корисний, коли користувачі мають чітко виражені потреби або завдання - як у випадку геометричних задач. Тут рекомендації базуються на логічних відношеннях між задачами, теоремами, визначеннями тощо [9].

Таким чином, для створення ефективної рекомендаційної системи у сфері освіти найкраще підходить підхід, що базується на структурованих знаннях. Він дозволяє формувати зрозумілі та обґрунтовані рекомендації, що особливо важливо у навчальному процесі. Така система не просто допомагає знайти

правильний розв’язок, а й підтримує розвиток логічного мислення та глибшого розуміння матеріалу.

Розділ 2. Аналіз предметної області та вибір онтологічної моделі

2.1 Особливості геометричних задач та вимоги до системи

Геометричні задачі мають характерну рису: їх належне розв’язання потребує лише одного - володіння набором означень та правил з можливістю ними оперувати. Зазвичай такі задачі передбачають наявність певного набору вхідних даних, на основі яких потрібно визначити невідомі величини. До типових задач можна віднести обчислення довжин сторін фігур, кутів, площ, об’ємів, відстаней між точками, а також комбіновані задачі, де присутні декілька взаємопов’язаних вимог. Обов’язковою умовою таких задач стає те, що вхідних тверджень повинно бути достатньо, щоб, поєднавши їх з відповідними правилами, які можуть бути застосовані у поставленому завданні, було фізично можливо віднайти невідому величину [6].

З цього випливають важливі вимоги до майбутньої рекомендаційної системи:

- підтримка різних типів геометричних об'єктів – як простих (точка, пряма, кут), так і складних (трикутник, чотирикутник тощо);
- можливість наслідування понять (наприклад, квадрат є окремим випадком прямокутника, а той – чотирикутника);
- підтримка властивостей та зв’язків між об’єктами (наприклад, “кут є прямим”, “трикутник має три вершини”) [6];
- наявність правил логічного виводу - система повинна мати змогу встановлювати нові факти на основі наявних тверджень [5].

Геометричні задачі мають специфічну природу: для їх розв’язання учню потрібно не просто знати формули, а розуміти суть геометричних понять і вміти логічно поєднувати відомі факти. Такі задачі зазвичай ґрунтуються на чітко визначених умовах, які, у поєднанні з відповідними властивостями фігур та

просторовими міркуваннями, дозволяють вивести шукані величини. Це можуть бути довжини, кути, площі або інші характеристики, і часто одна задача охоплює кілька взаємозалежних тверджень. Для коректного розв’язання важливо, щоб умови містили достатньо інформації для логічного виведення відповіді. Відповідно, рекомендаційна система, яка допомагає у вивченні геометрії, має відображати структуру геометричних знань, підтримувати різні рівні абстракції понять та бути здатною виводити нову інформацію, спираючись на вже відомі зв’язки та правила.

2.2 Бази знань

База знань - це структурована колекція інформації, система, розроблена для зберігання, організації та управління знаннями. Вона використовується для різних цілей, таких як: зберігання інформації у структурованому форматі (документація, інформація про продукт, покрокові інструкції), підтримка прийняття рішень, покращення ефективності роботи [5; 9]. Існують декілька моделей баз даних, а саме:

• Продукційна модель - у цій моделі знання представлені у вигляді “якщо (умова) – то (дія). Це дає можливість виводити нові знання з уже відомих фактів [5].

• Фреймова модель - це модель, в якій фрейм виступає як одиниця інформації для опису об’єктів та властивостей [5].

• Семантична модель - описує знання, використовуючи поняття, відношення та ієрархії [5; 9].

Для вирішення задачі кваліфікаційної роботи було обрано семантичну модель, яку представляють онтології.

2.3 Онтології і їх використання у побудові рекомендаційної системи

Онтологія - певне формалізоване представлення знань та фактів разом з їхніми властивостями та взаємозв’язками. Онтології допомагають структурувати знання у зрозумілій формі як для людей, так і для комп’ютерних систем [7; 8].

Сутність онтології позначають не чітко окреслену площину знань, а передбачає можливість розширення інформації спираючись на існуючі дані. Якщо описати коротко відмінність від звичайної бази даних та онтології, то база даних вважає істинними лише ті факти, які в ній містяться, в онтології ж відсутність інформації не доказує її хибності, а лише трактує її як таку, яку ще не довели. Основна перевага онтологій полягає у тому, що вони працюють за принципом відкритого світу: відсутність інформації не означає її хибності, що дозволяє системі гнучко розширювати знання у процесі її використання [7; 8].

На відміну від класичних реляційних баз даних, які оперують лише відомими фактами, онтологія дозволяє виводити нові знання на основі логічних правил, що є критично важливим для побудови системи рекомендацій [7; 9].

Онтологія зазвичай створюється шляхом визначення ключових сутностей і їхніх характеристик, а також встановлення взаємозв’язків між цими сутностями. Завдяки такій структурі формується розуміння різниці між об’єктами, що дає змогу робити висновки про нові, раніше неочевидні факти. Особливо важливо правильно спроектувати структуру онтології на початковому етапі, адже від цього залежить ефективність та надійність системи в подальшому.

Основні риси онтологій можна описати так:

- забезпечують структурованість ієрархії понять;
- дозволяють формалізувати логічні зв’язки між сутностями;
- є машиночитаними, що спрощує автоматичну обробку знань;
- підтримуються стандартами семантичного вебу [2; 3].

Саме ці риси роблять онтології ідеальним фундаментом для створення рекомендаційної системи у сфері розв’язку геометричних задач.

2.4 Структура онтології

Структура онтології включає в собі ієрархію об’єктів, логічні зв’язки між ними, разом з правилами та обмеженнями, які характеризують поведінку системи. Головна її особливість - це чіткість та однозначність в організації інформації. Варто детальніше розглянути ці визначення, оскільки саме вони складають основу онтології та без них вона втратила б свою суть [7; 8].

Тож, онтологія складається з таких ключових компонентів:

1. Сутності, вони ж об’єкти та поняття - елементарні або складні об’єкти предметної галузі. Вони можуть мати ієрархічну структуру по типу “предок - нащадок” та є основними складовими онтології. Вони визначають будь-що - об’єкти реального світу або абстрактні поняття. Прикладами сутностей є квадрат, кут, відрізок [7].
2. Класи - об’єднання сутностей зі спільними властивостями. Через класи реалізується наслідування та категоризація. Вони спрощують процес розробки, даючи змогу категоризувати сутності, а не оперувати ними по окремості. Прикладами класів є клас фігура, нащадками якого є класи трикутник і прямокутник [7; 8].
3. Зв’язки між сутностями. Вони можуть бути описані за допомогою словосполучень “є атрибутом”, “складається з” тощо. Вони позначають усі варіанти, як класи та сутності можуть бути зв’язані одні з одним. Зв’язки можуть містити власні характеристики, які будуть його уточнювати. Прикладом зв’язку є “трикутник складається з трьох відрізків і кутів” [7].
4. Властивості - характеристики сутностей. Вони потрібні для опису особливостей кожної з сутностей. Наприклад, “кут має 75 градусів” або “довжина сторони трикутника - 15 см” [7].

5. Логічні правила - певна формалізація відносини, створена з метою виводу нових фактів та правил на базі існуючих. Зазвичай описуються твердженням "якщо - то" для виявлення наслідків зв’язків між сутностями. Прикладом правила є твердження “якщо відношення сторін двох трикутників однакові - то ці трикутники є подібними” [5].
6. Аксіоми - правила, які накладаються на властивості або зв’язки між термінами. Вони формують логіку, якою буде оперувати онтологія. Наприклад, “Всі сторони рівностороннього трикутника мають однакову довжину” є аксіомою. [5]
7. Обмеження - це невід’ємна частина онтології. Вони потрібні для звуження предметної області та формалізації уточнень, необхідних для додавання нових властивостей, зв’язків та правил до онтології. Завдяки обмеженням можливо визначити які конкретні властивості притаманні класам та їх діапазон значень. Прикладом обмеження є “Гіпотенуза є стороною прямокутного трикутника, протилежною прямому куту”.

Всі ці поняття тісно взаємопов’язані і не можуть розглядатися окремо одне від одного. Процес створення онтології завжди починається з детального аналізу предметної області, що дозволяє максимально точно і коректно відобразити її сутність за допомогою відповідних структур і моделей. Від того, наскільки грамотно та продумано буде побудована структура онтології, залежить ефективність всієї системи загалом. Помилки чи неточності на цьому початковому етапі можуть призвести до логічних невідповідностей, втрати важливої інформації або складнощів у подальшому використанні онтології, що значно знизить її практичну цінність.

2.5 Описання онтологій та редактори для них

Онтології можуть бути зображені візуально у вигляді графів, але здебільшого по причині нечитабельності за умови оперування великими об’ємами

інформації, їх описують саме формальною мовою. Для формального опису використовується мова OWL (Ontology Web Language) [7; 8].

OWL - це мова, яка була створена з метою представлення інформації про окрему галузь та зв’язків у ній у семантичному вебі, у прямому перекладі - мова для опису онтологій. OWL побудована на стандартах RDF (Resource Description Framework), але розширює функціональність класів, властивостей та зв’язків, надаючи можливість описувати правила та обмеження у більш розгорнутому вигляді, що дозволяє моделювати складніші онтології. Завдяки цьому OWL стала потужним інструментом для побудови онтологій [7; 8].

Для створення та управління онтологіями використовують спеціалізовані редактори - програми, які значно спрощують процес розробки та підтримки онтологічних баз знань. Вибір конкретного редактора має велике значення, адже саме він визначає зручність роботи, наявність корисних функцій і можливостей для розширення, а також інтуїтивність інтерфейсу. Добре підібраний інструмент дозволяє не лише ефективно створювати структури онтологій, але й легко вносити зміни та інтегрувати додаткові модулі. Серед найбільш відомих редакторів, які користуються популярністю серед розробників, можна виділити такі, перелічені у порядку збільшення популярності:

- OntoWiki - інструмент для редагування онтологій з можливістю працювати над онтологією більше ніж одному користувачу одночасно, тому часто використовується у командній роботі [7].
- WebVOWL - сервіс, що використовується саме для візуалізації та вивчення вже готових онтології з графічним представленням її структури [7; 8].
- Protégé - лідер з інструментів для створення та редагування онтологій, який має зручний та інтуїтивно зрозумілий графічний інтерфейс, підтримує такі стандарти семантичного вебу, як OWL, RDF та надає широку функціональність завдяки великої кількості додаткових плагінів.

Інструмент постійно розвивається та оновлюється, завдяки чому не втрачає свою першість на ринку вже багато років [7; 8].

Для розробки рекомендаційної системи було обрано редактор Protégé через його багатий набір функцій, інтуїтивно зрозумілий інтерфейс і наявність вбудованих інструментів, які значно полегшують процес створення та управління онтологіями. Цей редактор широко використовується в наукових та прикладних проектах завдяки своїй гнучкості, підтримці різних форматів і можливості розширення за допомогою плагінів, що робить його ідеальним вибором для реалізації складних систем на базі онтологій.

Розділ 3. Обробка природної української мови

Наша майбутня система передбачає не тільки вирішення задач та рекомендацій щодо їх вирішення для учнів, а й у першу чергу розпізнати задачу та її елементи, яка передана до системи у вигляді природної мови. Цей розділ саме про те, як використовується мова, про існуючі методи її обробки та вибір одного з них для реалізації.

3.1 Проблематика

Мова є основним засобом комунікації між людьми. Природні мови - це ті, що виникли еволюційно в процесі людського спілкування, на відміну від формальних мов, які створені штучно, зокрема для взаємодії з комп’ютерами. Однією з форм таких формальних мов є мови програмування - вони служать інструментом для передавання інструкцій від людини до машини.

Природні мови, зокрема українська, надзвичайно багаті та складні. Вони мають неоднозначності на різних рівнях: синтаксичному, семантичному, морфологічному та референційному. Саме ці особливості ускладнюють їхнє безпосереднє використання для спілкування з комп’ютером.

Обробка природної мови (англ. *Natural Language Processing*, NLP) - це міждисциплінарна галузь, яка поєднує лінгвістику, інформатику, машинне навчання та штучний інтелект. Метою NLP є створення моделей і алгоритмів, які дозволяють комп’ютерам аналізувати, інтерпретувати та ефективно взаємодіяти з текстами або висловлюваннями, написаними природною мовою.

Хоча сучасні системи ще не досягли рівня повного «розуміння» мови, наближеного до людського, вже існують ефективні інструменти для аналізу тексту, автоматичного перекладу, розпізнавання мовлення та генерації природномовного тексту. Втім, більшість таких систем орієнтована переважно на англійську мову, яка має статус глобальної мови міжнародного спілкування та

наукової комунікації [1].

У цій кваліфікаційній роботі основна увага приділяється обробці текстів саме українською мовою. Це додає певних викликів, адже сучасні технології обробки природної мови (NLP) для української ще не настільки розвинені, як для англійської чи інших поширених мов. На відміну від англомовного сегменту, де існує безліч готових інструментів, бібліотек і моделей, для української мови кількість таких рішень досі є обмеженою. Це створює додаткові складнощі під час розробки комплексних систем аналізу тексту, адже доводиться витрачати більше ресурсів на адаптацію, налаштування або навіть власну розробку необхідних компонентів. Водночас, це відкриває перспективи для створення нових інструментів і покращення якості NLP для української мови.

3.2 Методи та підходи в обробці природної мови

Обробка природної мови (NLP) охоплює широкий спектр методів, які дають змогу комп’ютерам аналізувати та інтерпретувати текст, написаний звичайною людською мовою. У межах цієї роботи використовуються ключові інструменти NLP, необхідні для попередньої обробки українського тексту, вилучення смислових одиниць та структурування даних для подальшої обробки.

Токенізація - це процес поділу тексту на окремі одиниці, які мають лінгвістичне значення: слова, речення або абзаци. В українській мові поділ на слова здебільшого базується на використанні пробілів. Натомість визначення меж речень є складнішим завданням, адже потребує врахування абревіатур, імен власних і різноманітних пунктуаційних винятків.

Лематизація - це метод зведення слова до його базової словникової форми (леми), що враховує морфологічний аналіз. На відміну від стемінгу, лематизація забезпечує значно вищу точність та зберігає семантичну відповідність між формами слова. У цій роботі застосовується саме лематизація як ключовий етап

підготовки тексту до подальшого аналізу.

Розмічування частин мови (POS-tagging) - процес визначення граматичної ролі кожного слова в тексті (іменник, дієслово, прикметник тощо). Ця розмітка є важливою для розуміння синтаксичних зв’язків між словами та для точнішої інтерпретації смислу. У випадку української мови POS-розмітка ускладнюється багатозначністю окремих слів залежно від контексту.

Розпізнавання іменованих сутностей (Named Entity Recognition, NER) - метод виявлення в тексті конкретних типів інформації, як-от імена людей, географічні назви чи математичні об’єкти. У контексті геометричних задач це може стосуватись розпізнавання назв фігур, чисел, одиниць виміру тощо.

Видобування інформації (Information Extraction) - процес автоматичного виявлення структурованих даних із не структурованого тексту. Це дозволяє виявити сутності, встановити зв’язки між ними, та побудувати формалізоване представлення задачі для подальшої обробки або рекомендації.

Застосування вищенаведених методів дозволяє трансформувати текстову форму геометричних задач у структури, які зручно обробляти в рамках онтологічних баз знань. Саме така трансформація є критично важливою для ефективної реалізації рекомендаційної системи, орієнтованої на задачі шкільного рівня [1].

3.3 Використання мовного аналізатору

Для реалізації більшості методів та підходів, описаних у підрозділі 2.2, необхідне використання морфо-синтаксичних аналізаторів. У сучасному програмному забезпеченні вже існують готові рішення такого типу. Проте варто зазначити, що кількість якісних мовних аналізаторів для української мови значно менша, ніж для англійської. Це пов’язано з тим, що більшість методів обробки

природної мови (NLP) вимагають наявності спеціалізованих текстових корпусів [1;10].

Текстовий корпус - це впорядкована та ретельно зібрана колекція текстів певною мовою. Найціннішими вважаються корпуси з анотацією - тобто ті, де до кожного слова (токена) додано морфологічну інформацію: рід, число, відмінок тощо. Такі корпуси можуть бути розмічені вручну або автоматично. Ручна розмітка здійснюється фахівцями-науковцями й, як правило, забезпечує вищу точність, ніж автоматична. Однак, через високу трудомісткість, кількість вручну розмічених корпусів для будь-якої мови обмежена [10].

З огляду на це, для цієї роботи було обрано інструмент UDPipe, який відзначається високою точністю та підтримкою української мови. UDPipe є навчальним конвеєром для обробки текстів, що виконує токенізацію, лематизацію, визначення частин мови, морфологічний аналіз і побудову синтаксичних залежностей [11].

UDPipe базується на концепції універсальних залежностей (Universal Dependencies або UD) - міжнародного проєкту, що має на меті створення єдиної метамови для опису синтаксичних зв’язків у природних мовах. Основою UD є теорія синтаксичних залежностей, згідно з якою кожне слово в реченні пов’язане з іншим за принципом “голова–залежне”. Цей зв’язок зазвичай візуалізується стрілкою від головного до залежного слова [12]. UDPipe підтримує обробку текстів 72 природними мовами.

Функціональні можливості UDPipe включають:

- поділ тексту на токени;
- лематизацію;
- визначення частин мови;
- морфологічний аналіз;

- визначення головного слова для кожного токена (ідентифікатор або нуль);
- встановлення синтаксичного зв’язку (UD-відношення) між токеном і його головою (або з коренем речення, якщо голови немає) [11].

Слід також відзначити, що UDPipe має елементи розпізнавання іменованих сутностей (NER): наприклад, у процесі морфологічного розбору імен та прізвищ програма може вказати, що слово є власною назвою [11].

Для української мови була створена модель Universal Dependencies (UD), яка навчена на спеціально підготовленому корпусі текстів - так званому "золотому" стандарті. Цей корпус містить тексти, які були вручну розмічені експертами на кількох рівнях: від морфології (форма слова, частина мови) до синтаксису (структура речення і взаємозв’язки між словами). Такий багаторівневий і ретельний підхід до анотації текстів забезпечує високу точність і надійність мовного аналізу, що дуже важливо для подальшого використання моделі в різноманітних завданнях обробки природної мови.

Розділ 4. Теоретичні аспекти геометричних задач у онтологічних базах знань

4.1 Вибір предметної області

Як уже зазначалося раніше, основною метою цієї кваліфікаційної роботи є розробка рекомендаційної системи, яка могла б допомогти у вирішенні математичних задач. Така система має вміти самостійно аналізувати умови задачі, генерувати план розв’язання та надавати кінцеву відповідь.

У якості цільової предметної області було обрано саме геометричні задачі. Це пояснюється тим, що геометрія характеризується чіткою структурою, формальними правилами та відносною передбачуваністю процесу розв’язання, що суттєво полегшує автоматизовану обробку та моделювання розв’язків.

Поняття «математична задача» є надзвичайно широким і включає задачі з різних галузей математики: алгебри, комбінаторики, теорії чисел тощо. Однак для ефективної розробки та тестування системи необхідно обмежити множину задач певною вузькою категорією, щоб забезпечити більш цілеспрямований підхід і кращу якість рекомендацій [6]. Геометрія, як дисципліна, що активно використовує візуальні, формальні та логічні елементи, стала ідеальним кандидатом для цієї мети.

Цей вибір також обґрунтовується тим, що у геометрії добре розвинені формальні описи об’єктів і відносин, що створює сприятливі умови для побудови онтологій та семантичних моделей, які лежать в основі сучасних рекомендаційних систем [5].

4.2 Типологія геометричних задач

Геометричні задачі мають різні формати представлення залежно від їхньої природи та контексту. У науковій і освітній літературі можна виокремити кілька

основних типів геометричних задач:

- Задачі на побудову. Такий вид задач вимагає побудови геометричних фігур з використанням інструментів креслення (лінійки, циркуля, транспортира). Приклад: побудова трикутника за заданими сторонами, проведення бісектриси, побудова дотичної до кола.
- Задачі на доведення. Їхня суть полягає у тому, що вказується певне твердження, яке потрібно довести, використовуючи логічні міркування та відомі геометричні властивості. Ці задачі перевіряють знання теорії та здатність її застосовувати. Учень має вказати чітку послідовність дій та маніпуляцій, які потрібно реалізувати, щоб однозначно підтвердити або заперечити певну тезу. Приклад: довести, що діагоналі прямокутника рівні.
- Задачі на обчислення. Такі задачі передбачають пошук числових значень геометричних величин (кутів, сторін, площ, периметрів тощо) на основі заданих умов. Задачі вимагають володіння формулами та вміння використовувати характеристики фігур. Ці задачі найкраще піддаються алгоритмізації, оскільки містять чітко визначені математичні залежності. Вони розвивають аналітичне мислення для пошуку правильного алгоритму розв'язання. Приклад: знаходження кутів, периметру, площі, невідомих сторін фігур, відстаней між точками тощо.
- Комбінаторні задачі. Вони включають нестандартні, часто логічні задачі, які не завжди мають однозначне або очевидне рішення. Ці задачі спрямовані переважно на розвиток логічного та творчого мислення та аналізу. Вони важко піддаються типізації, оскільки кожна задача сформульована унікальним чином. Така структура ускладнює автоматизацію процесу пошуку алгоритму для розв’язку. Прикладом буде задача на знаходження кількості діагоналей у багатокутнику.

Для побудови автоматизованої рекомендаційної системи в межах онтологічної бази знань найбільш доцільними є задачі на обчислення. На відміну

від задач на побудову, які потребують візуального представлення, та задач на доведення з їх складною логікою, задачі на обчислення базуються на чітких правилах і формульних залежностях. Ці характеристики полегшують формалізацію знань у вигляді фактів, аксіом і правил, що є основою онтологій [5].

Комбінаторні задачі через високу варіативність умов і відсутність стандартизованих методів розв'язання значно ускладнюють автоматизацію і тому не були обрані для розгляду в межах цієї роботи.

Таким чином, задачі на обчислення було обрано як основний тип задач для реалізації в рамках кваліфікаційної роботи через їх формалізованість та придатність до автоматизованої обробки.

Для конкретизації тематики, обрана вибірка задач, які включають:

- визначення типу фігури (рівнобедрений, прямокутний чи рівносторонній трикутники);
- визначення типу кутів;
- обрахування градусної міри невідомих кутів та довжин сторін;
- знаходження периметра та площі фігури;

4.3 Приклад формалізації геометричної задачі

Для ілюстрації підходу розглянемо приклад задачі з підручника "Мерзляк А. Г. Геометрія: підруч. для 7 кл. закладів заг. серед. освіти." [6]:

200.° Периметр рівнобедреного трикутника дорівнює 28 см, а бічна сторона — 10 см. Знайдіть основу трикутника.

Рис 4.3.1. Скриншот з підручника

План розв'язання даної задачі:

1. Сформулювати перелік вхідних даних у вигляді набору однозначних

тверджень разом з величиною яку необхідно знайти:

- Трикутник є рівнобедреним.
- Бічна сторона трикутника дорівнює 10 см.
- Периметр трикутника дорівнює 28 см.
- Шукана величина – основа трикутника.

2. Виділити набір правил, які стосуються цієї задачі:
   - У рівнобедреному трикутнику бічні сторони рівні.
   - Периметр трикутника - це сума довжин усіх його сторін.
3. Пошук невідомих величин.
   - З перших двох тверджень та правил, які з ними пов’язані, можна визначити другу бічну сторону трикутника: довжина другої сторони дорівнює даній довжині першої, отже дорівнює 10 см.
   - Одне з правил про периметр і саме його ми використаємо для знаходження основи. Сума довжин відомих (тобто бічних) сторін трикутника дорівнює 10 см + 10 см = 20 см.
   - Тоді можемо знайти основу, віднявши від периметра суму довжин двох інших сторін, яку ми щойно знайшли. Тож, основа дорівнює 28 см - 20 см = 8 см.
4. Виведення відповіді, яку потрібно записати твердженням, що відповідає на питання задачі. В даному випадку таким твердженням є “Основа трикутника дорівнює 8 см.”

Цей приклад демонструє, як задачі можуть бути представлені у вигляді набору тверджень (фактів), які у поєднанні з правилами дозволяють за допомогою логічного виводу отримати відповідь. Саме такий формат є придатним для представлення в онтологічній базі знань, де факти і правила зберігаються у вигляді трійок (суб’єкт - предикат - об’єкт), а механізм логічного виводу (reasoning) дозволяє знаходити нові знання.

Розділ 5. Реалізація алгоритму

5.1 Розробка онтології в Protege

Головним компонентом розробленої системи стала онтологія, яка використовує правила SWRL разом з аксіомами. Перш за все, необхідно було продумати коректну класову структуру для неї, яка визначала б базові геометричні поняття, взаємозв’язки та залежності між сутностями та надавала можливість уточнювати надані дані і автоматично виводити нові твердження.

Основними вимогами для класів є:

1. Узагальненість та зрозумілість. Ієрархія має охоплювати всі можливі геометричні об’єкти
2. Формалізація відношень між об’єктами, наприклад, "є стороною", "лежить на", "є частиною"
3. Розширюваність - можливість додати нові класи та зв’язки між ними не змінюючи існуючу структуру

Отже, було обрано наступну ієрархію класів:

1. Точка. Це фундаментальне поняття у геометрії, яке неможливо визначити через інші. Кожен елемент так чи інакше складається з точки.
2. Відрізок. Наслідується від точки, оскільки сам являє собою набір точок. Є можливість спрощено його представити як клас, який характеризується двома точками - його кінцями, порядок яких не має враховуватись. Це пояснюється тим, що зазвичай відрізок і позначається двома великими буквами латинського алфавіту, наприклад AB, де A та B - точки, при тому що AB та BA - це один і той самий відрізок. Відрізок має властивість довжини, вираженої в сантиметрах.
3. Кут. Це клас, що складається з трьох точок, середня з яких обов’язково вказує на вершину кута. Оскільки поняття напрямленого кута не є

необхідною умовою в контексті системи, що розглядається, порядок двох інших крайніх точок будемо вважати як не важливий, аналогічно з відрізком. Даний клас повинен мати таку властивість, як міра кута, яка є дійсним числом та визначається у градусах.

4. Класи, що характеризують підвиди кута, а саме гострий, тупий та прямий кути.
5. Фігура. Оскільки кожна існуюча двовимірна фігура має периметр та площу, доцільно створити окремий абстрактний клас, який буде володіти цими властивостями. У такому випадку, кожна фігура буде наслідуватись цими характеристиками.
6. Трикутник. Цей клас наслідує клас Фігура разом з його властивостями.
7. Класи, що характеризують підвиди трикутника. За властивостями кутів та сторін, можна виділити наступні шість класів. За видом кутів вони поділяються на тупокутні, гострокутні та прямокутні, за видом сторін - на рівносторонні, рівнобедрені та різносторонні трикутники. Один і той самий трикутник завжди підпадатиме під два класи одночасно.
8. Промінь. Це частина прямої лінії, яка має початок, але не має кінця.
9. Пряма. Це лінія, яка не має ні початку, ні кінця.
10. Чотирикутник та класи що описують його підвиди, такі як квадрат, паралелограм, ромб.
11. Еліпс та його підклас коло.

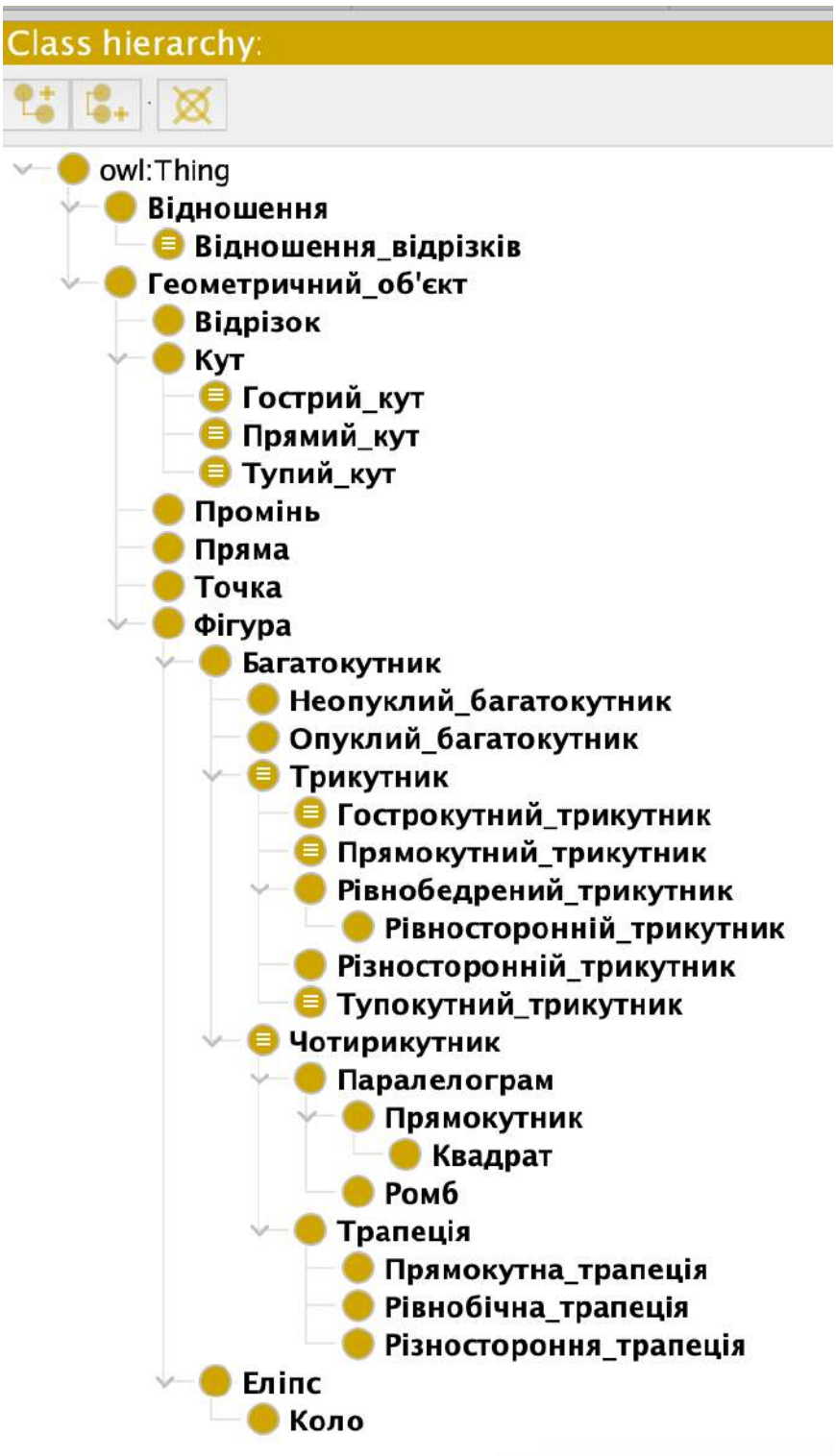


Рис 5.1. Ієрархія класів у Protege

Наступним кроком стало визначення властивостей для даних класів. У Protege властивості бувають трьох різних типів: Object Properties, Data Properties та Annotation Properties.

Object Properties, або властивості об’єктів - це усі характеристики, які стосуються класів та визначають їхні взаємозв’язки між собою.

Data Properties призначені для того, щоб мати змогу описувати усі дані через конкретні числові або текстові значення.

Annotation Properties створюються для додавання метаданих.

На рисунку зображено Object Properties для нашої моделі:

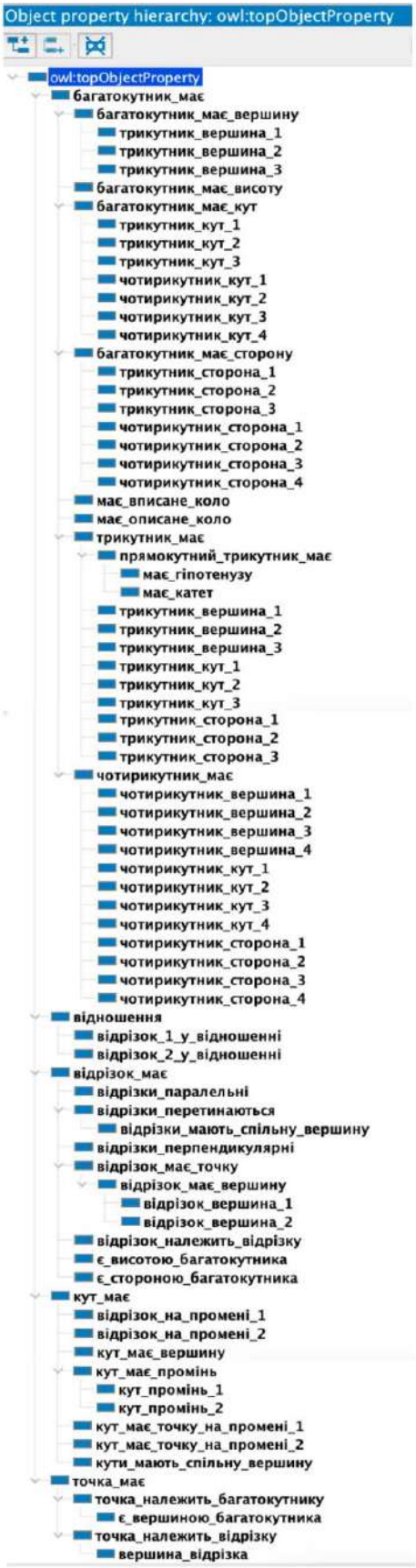


Рис 5.2. Об’єктні властивості у Protege

На наступному рисунку зображено Data Properties:

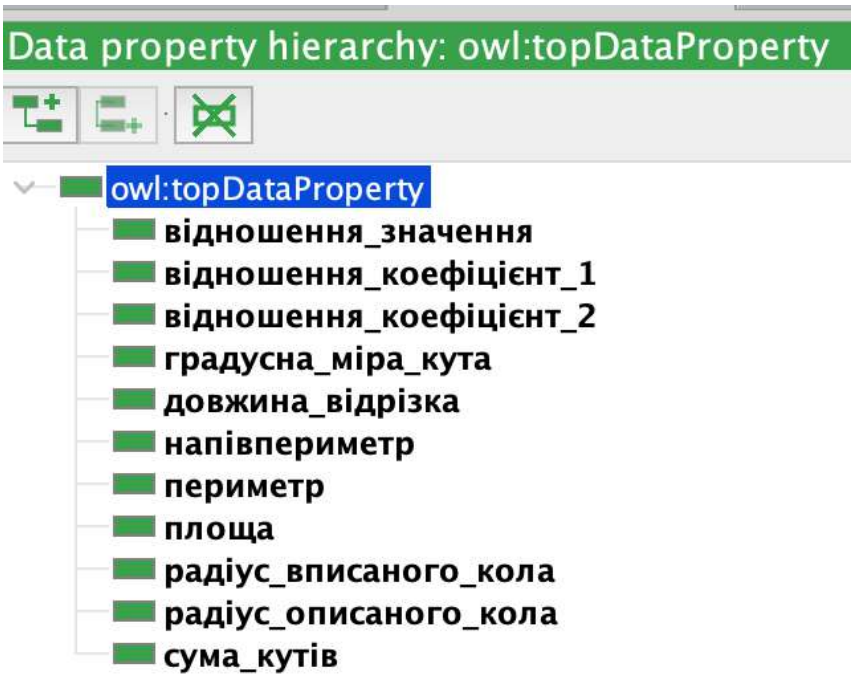


Рис 5.3. Data-властивості у Protege

Їх опис:

1. відношення_значення - встановлює числове значення (типу double) відношення для класу Відношення
2. відношення_коефіцієнт_1 - встановалює перший коефіцієнт відношення (типу double) для класу Відношення_відрізків
3. відношення_коефіцієнт_2 - встановалює другий коефіцієнт відношення (типу double) для класу Відношення_відрізків
4. градусна_міра_кута - встановлює значення кута в градусах (типу double) для класу Кут
5. довжина_відрізка - встановлює довжину відрізка (типу double) для класу Відрізок
6. напівпериметр - встановлює значення половини периметру фігури (типу double) для класу Фігура
7. периметр - встановлює значення периметру фігури (типу double) для класу

Фігура

8. площа - встановлює значення площі фігури (типу double) для класу Фігура
9. радіус_вписаного_кола - встановлює радіус вписаного кола (типу double) для класу Фігура
10. радіус_описаного_кола - встановлює радіус вписаного кола (типу double) для класу Фігура
11. сума_кутів - встановлює суму кутів фігури (типу double) для класу Фігура

Для доведення та визначення результатів і висновків системи розроблено набір SWRL правил, що реалізують геометричні аксіоми, теореми, формули та правила.

Рис 5.4. SWRL правила

5.2 Аналіз тексту

У рамках дослідження реалізовано модуль для виокремлення семантично значущої інформації з тексту українською мовою, що описують сутності геометричних об’єктів та їхні властивості. Ядром цього модуля є клас довільна навчена модель UDPipe для лематизації та аналізу тексту, виявлення зв’язків отриманих токенів.

Так як обробка природної мови не надає нам повне розуміння про структуру заданих об’єктів, реалізовано компонент для визначення важливих для вирішення сутностей та їх властивостей «GeometryExtractor».

```
def extract_chain(self, text):
    parsed = self.parser.analyze_text(text)
    sentences = self.parser.parse_conllu(parsed)
    return self.extract_entity_property_value_chain(sentences)
```

Рис 5.5. Метод виокремлення об’єктів онтології

Функціонування «GeometryExtractor» забезпечується кількома допоміжними класами:

- «UdipipeParser» - інкапсулює запуск мовної моделі та обробки тексту на основі бібліотеки UDPipe. Він забезпечує попередній лінгвістичний аналіз вхідного тексту, такий як токенізація, лематизація, морфологічне тегування та побудову залежностей між словами у форматі CoNLL-U.

- «EntityPropertyResult» - зберігає визначену сутність(геометричну фігуру), її властивості та пов’язані з нею індивіди у вигляді словника класів та виокремлених властивостей та зв’язків.

```python
class EntityPropertyResult:
    def __init__(self, individual, properties):
        self.individual = individual
        # Ensure every value is a list
        self.properties = {
            prop: vals if isinstance(vals, list) else [vals]
            for prop, vals in properties.items()
        }
```

Рис 5.6. Клас результуючої структури

Даний модуль виконує семантичний розбір тексту з метою ідентифікації геометричних об'єктів (наприклад, «трикутник», «коло») та значень їх числових або об’єктних властивостей (наприклад, «площа», «радіус», «катет»). Реалізація алгоритму має попередньо зазначені словники за яким здійснюється пошук лем і співставлення їх з онтологічною базою. Дана структура зберігає дані про існуючі класи, їх об’єктні властивості та можливі числові значення і може бути необмежено розширена для пошуку нових структур. Алгоритм проходить декілька етапів:

1. **Мовна обробка тексту:** вхідний текст обробляється і перетворюється у список токенів, розділених на списки токенів.
2. **Пошук фігури:** для кожного речення здійснюється пошук леми, що відповідає певній геометричному об’єкту(наприклад, «трикутник») за допомогою окремого словника співставлень.
3. **Отримання властивостей:** для кожної знайденої сутності проводиться пошук попередньо зазначених властивостей або пов’язаних індивідів. Якщо така властивість виявлена, здійснюється пошук пов’язаних числових значень (наприклад, «радіус 5»).
4. **Формування результату:** знайдені сутності зберігаються в списку об’єктів типу «EntityPropertyResult», який структурує знайдені дані для подальшої інтеграції з онтологічною системою.

Запропоноване рішення дозволяє автоматично виділяти сутності та їх опис

із сирого тексту, що є необхідним етапом при формуванні фактографічних баз знань та рекомендаційних систем.

5.3 Взаємодія з онтологією

Даний модуль реалізує програмну інтерфейсну обгортку для взаємодії з OWL-онтологією, використовуючи бібліотеку «owlready2». Його функціональність поділяється на два основні класи - «Ontology» та «OntologyInjector».

Створено клас «Ontology», який реалізує базові методи взаємодії з онтологією , працює з RDF та OWL онтологіями для зберігання та обробки геометричних даних.

Основними функціями класу є:

- Додавання індивідів:
- Маніпуляції з властивостями
- Отримання даних індивідів онтології

Клас «OntologyInjector» є допоміжним інструментом для додавання індивідів та всіх їх властивостей у вигляді структурованих об’єктів із властивостями до онтології.

```
class OntologyInjector:
    def __init__(self, ontology):
        """
        Ініціалізація інжектора для роботи з онтологією.
        :param ontology: Об'єкт онтології.
        """
        self.ontology = ontology
        self.counter = count()
```

Рис 5.7. Властивості класу «OntologyInjector»

Метод «inject» використовує OWL-api, щоб визначити класи та типи властивостей й ініціює їх додавання за описом та запускає рекурсивний механізм додавання атрибутів.

```
def inject(self, entity_property_results):
    """
    Інжектує дані в онтологію.
    :param entity_property_results: Список результатів з властивостями.
    """
    for i, descr in enumerate(entity_property_results, start=1):
        individual_name = f"{descr.individual}{i}"
        main_individual = self.ontology.add_individual(descr.individual, individual_name)
        self.add_properties_recursive(main_individual, descr.properties, individual_name)
```

Рис 5.8. Механізм інжекції індивідів

Завдяки гнучкій архітектурі та підтримці як data-, так і object-властивостей, цей модуль дозволяє легко переносити результат мовного чи логічного аналізу у існуючу базу знань, зберігаючи логічні зв’язки між індивідами онтології. Це робить його придатним для застосування в інтелектуальних рекомендаційних системах для розв’язування задач, зокрема у сфері геометрії або семантичного моделювання знань.

Роль інтеграції з механізмами логічної обробки фактів виконує - клас «ReasonerRunner». Він забезпечує взяємодії з різонером та автоматизовану обробку виведених ним з наявної онтології результатів.

Основними завданнями цього компоненту є ініціалізація процесу міркування над отриманими даними онтології з використанням правил SWRL (Semantic Web Rule Language), також відбір та форматування пояснень для покрокового виведення користувачу. Під час виклику функцій цього класу викликається OWL-сумісний різонер Pellet, виконується валідація введених та шуканих даних, фіксується послідовність доказів та логічних кроків, що ведуть до отримання результату. Виконується після обробка отриманих даних та форматування їх до зрозумілого уніфікованого представлення.

Так як owlapi для python не дає змоги запускати різонер Pellet, було створено окрему збірку на Java. Вона має один метод, що приймає шлях до онтології індивіда та властивість значення якої треба розрахувати та пояснити.

```
public static void getLogValues(String file, String individualName, String propertyName) throws Exception {  1 usage
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    OWLOntology ontology = manager.loadOntologyFromOntologyDocument(new FileDocumentSource(new File(file)));
    OWLDataFactory factory = manager.getOWLDataFactory();
    OpenlletReasoner reasoner = OpenlletReasonerFactory.getInstance().createReasoner(ontology);

    PelletExplanation explanation_generator = new PelletExplanation(ontology,  useGlassBox: false);
```

Рис 5.9. Метод збірки для висновування

Підсумовуючи, «ReasonerRunner» є ланкою що поєднує семантичне представлення даних в онтології із користувацьким інтерфейсом, що відображає процеси розв'язання завдань через роботу з послідовними кроками. Розроблений компонент надає обширний та зручний доступ для взаємодії з онтологією.

5.4 Взаємодія з користувачем

В даній роботі також було реалізовано графічний інтерфейс користувача (GUI) засобами бібліотеки «Tkinter». Інтерфейс слугує засобом взаємодії користувача з інтелектуальною системою розв'язування задач на основі онтологічного підходу та забезпечує інтуїтивно зрозумілий ввід задачі та відображення результатів.

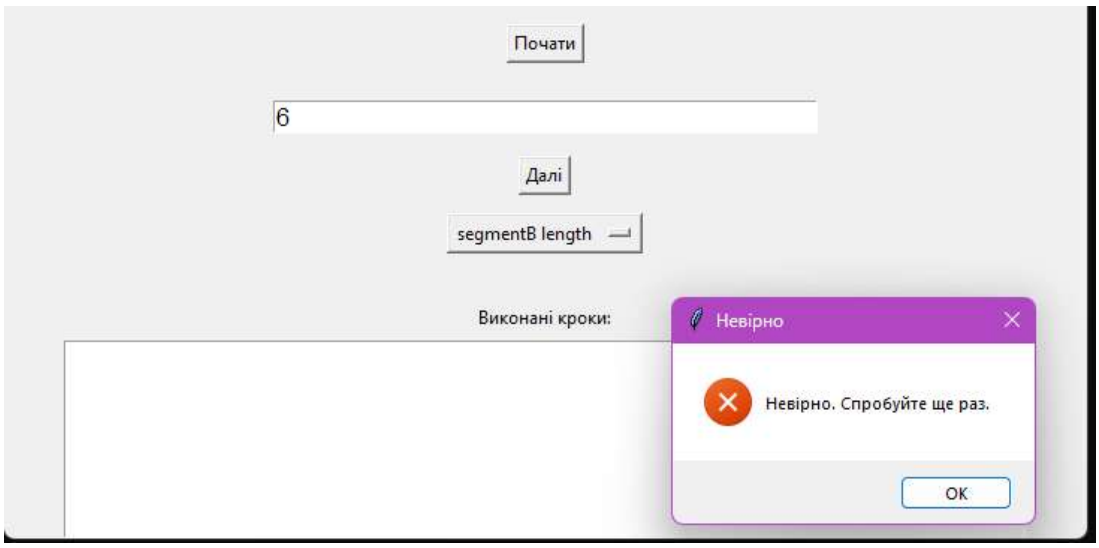


Рис 5.10. Перевірка кроку рішення користувача

Створений інтерфейс інкапсулює взаємодію користувача з семантичним і онтологічним ядрами системи за допомогою зовнішніх модулів, що забезпечують аналіз тексту, автоматичне визначення сутностей та запуск логічного висновування, що дозволяє його повторне використання або ж розширення для вирішення будь-яких задач на основі природного тексту.

Після введення та аналізу умови забезпечується покрокова перевірка рішень користувача, де кожен етап є результатом логічного висновування й порівнюється з введеним користувачем значенням і обраною дією. Графічний компонент також забезпечує відображення пройдених етапів та розрахунків у межах поточного сеансу разом з візуальним контролем ходу розв’язання, що забезпечує зручність сценарію та вибір наступних етапів.

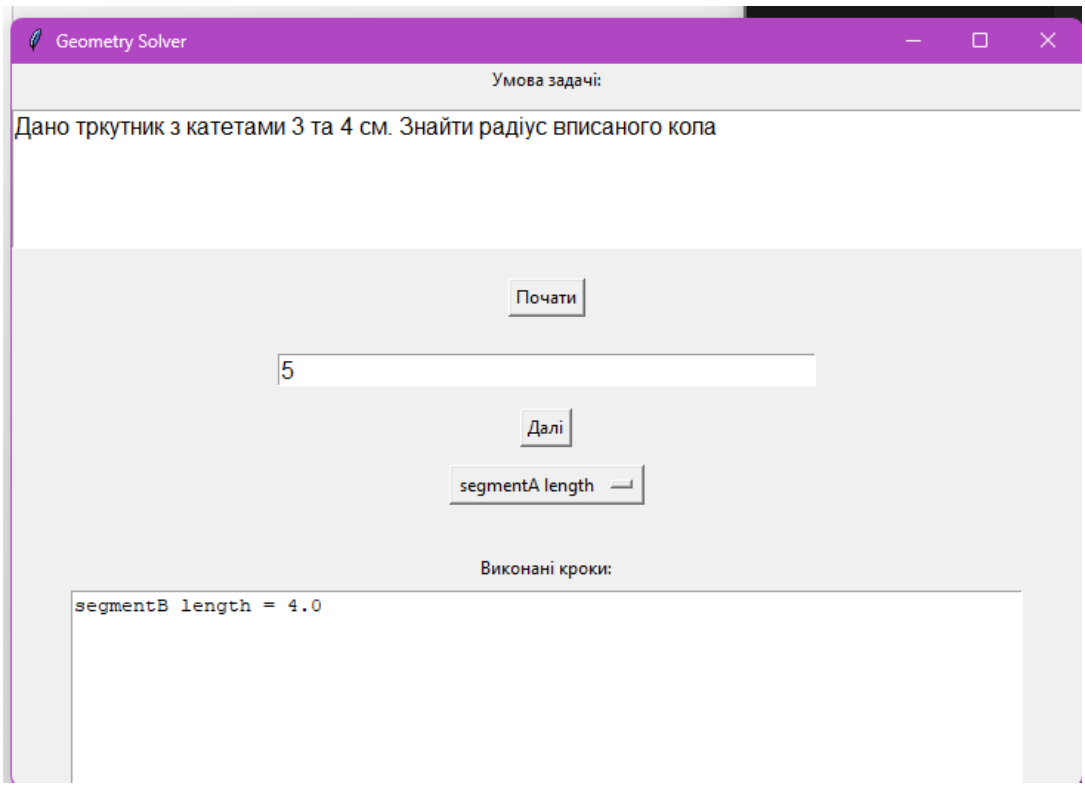


Рис 5.11. Інтерфейс користувача

Графічний інтерфейс інтуїтивно зрозумілий доступ до розробленого

механізму обробки даних і має змогу динамічно адаптуватися до змін задач та середовища. Такий підхід дозволяє легко замінити реалізації внутрішніх компонентів системи для використання в інших предметних областях, або ж альтернативних реалізацій механізмів, без необхідності додаткових доопрацювань.

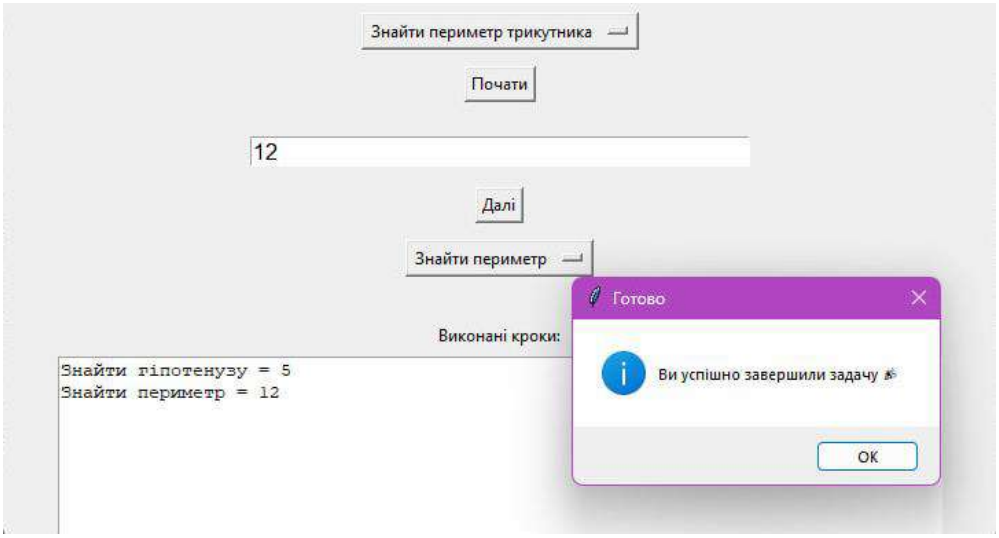


Рис 5.12. Підтвердження виконання завдання

5.5 Приклади реалізації та розв’язання обраних задач

Умова 1: З вершини тупого кута B паралелограма ABCD проведено висоту BK. Сторона BK відноситься до AB як 1/2. Знайдіть кут D паралелограма.

Анадіз задачі:

1. ABCD - паралелограм.
2. AB - одна зі сторін (основа).
3. BK - перпендикуляр (висота) з вершини B на сторону AD або продовження AD.
4. B - тупий кут.
5. A - гострий кут.
6. BK =1/2 AB

Властивості паралелограма:

1. Протилежні сторони рівні: AB = CD, AD = BC.
2. Протилежні кути рівні: A= C, B = D.
3. Сума сусідніх кутів = 180°.

Після введення умови в графічному інтерфейсі, починається мовний аналіз тексту та визначення індивідів за допомогою співставлення лем:

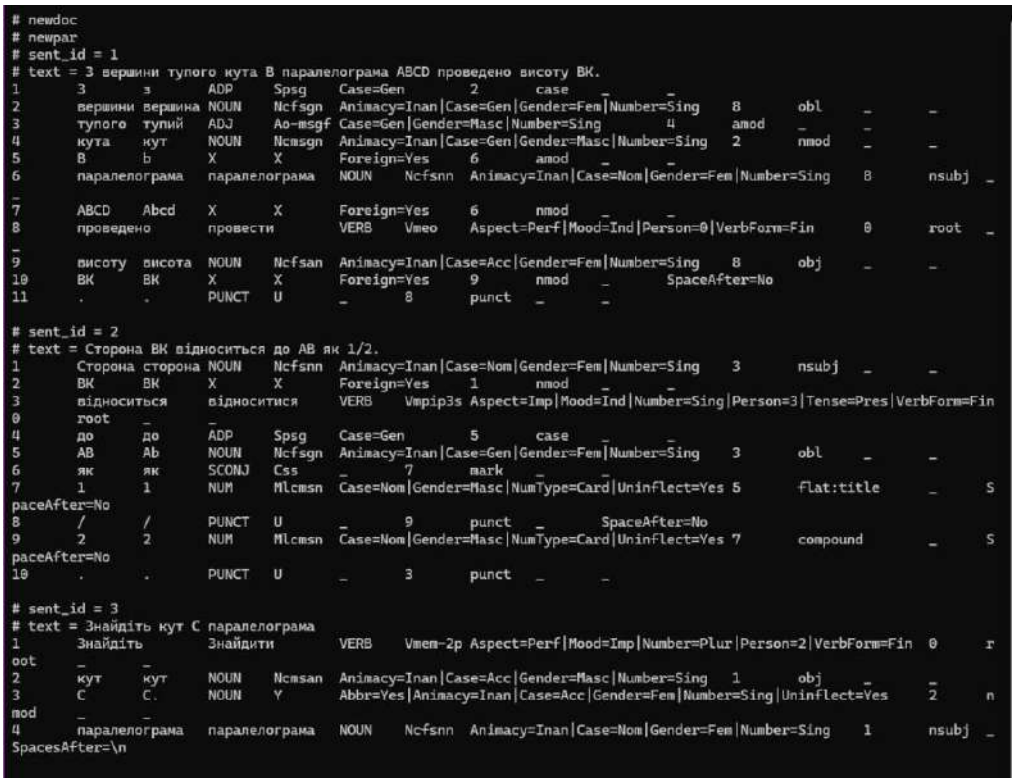


Рис 5.13. Результат лематизації

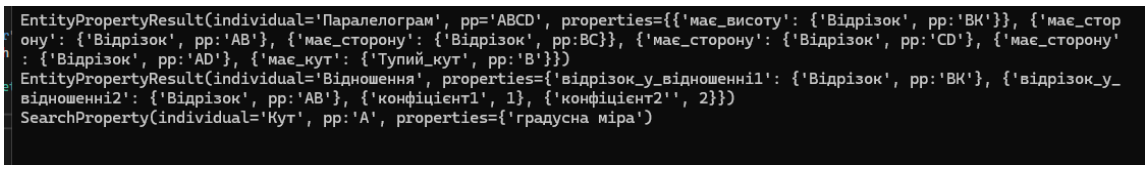


Рис 5.14. Результат визначення індтвідів

Далі всі індивіди завантажуються в онтологію і починається висновування всіх знайдених результатів. Основні SWRL правила, що необхідні для

розв’язання даної задачі:

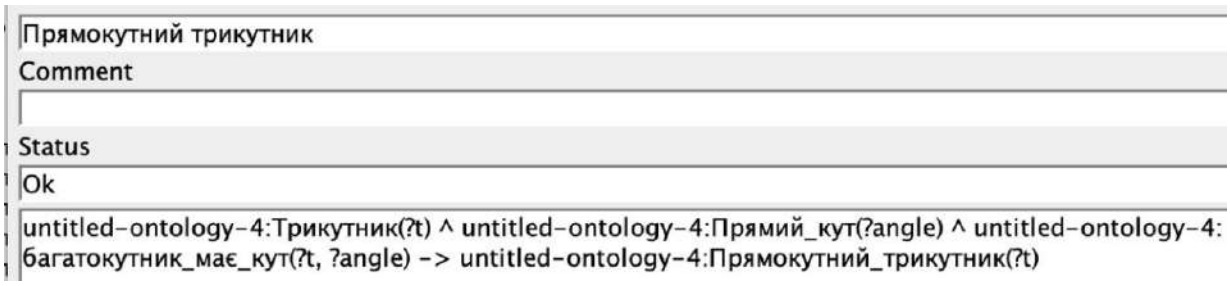


Рис 5.15. SWRL-правило. Прямокутний трикутник

Дане правило визначає, що якщо у трикутника є прямий кут, то він прямокутний.

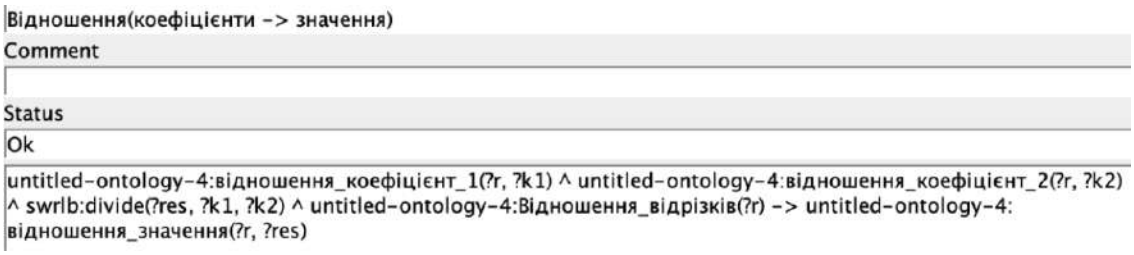


Рис 5.16. SWRL-правило. Відношення сторін

Наступні правила перебирають всі можливі сторони для катетів та гіпотенузи та виводять, що кути у прямокутному трикутнику дорівнюють 30 і 60 градусів, якщо один з катетів дорівнює половині гіпотенузи.

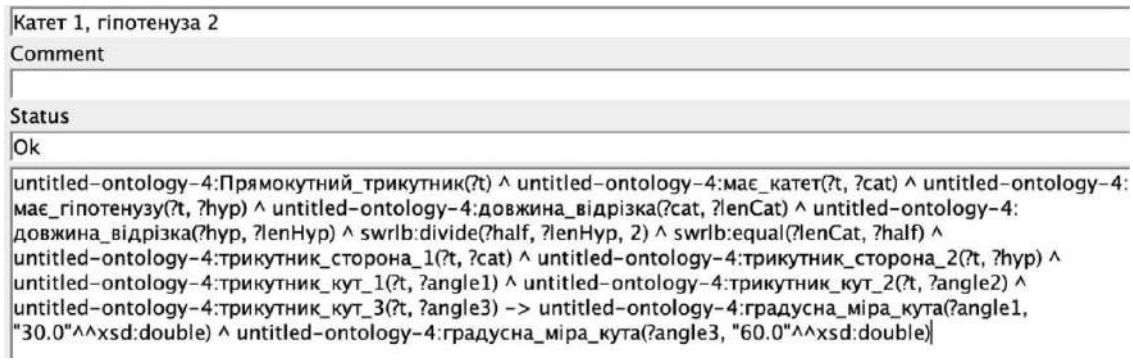


Рис 5.17. SWRL-правило. Катет 1. Гіпотенуза 2

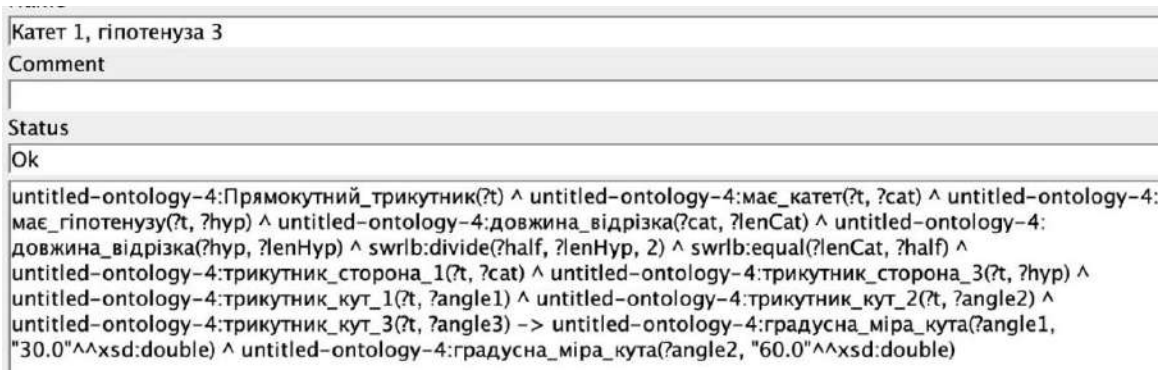


Рис 5.18. SWRL-правило. Катет 1. Гіпотенуза 3

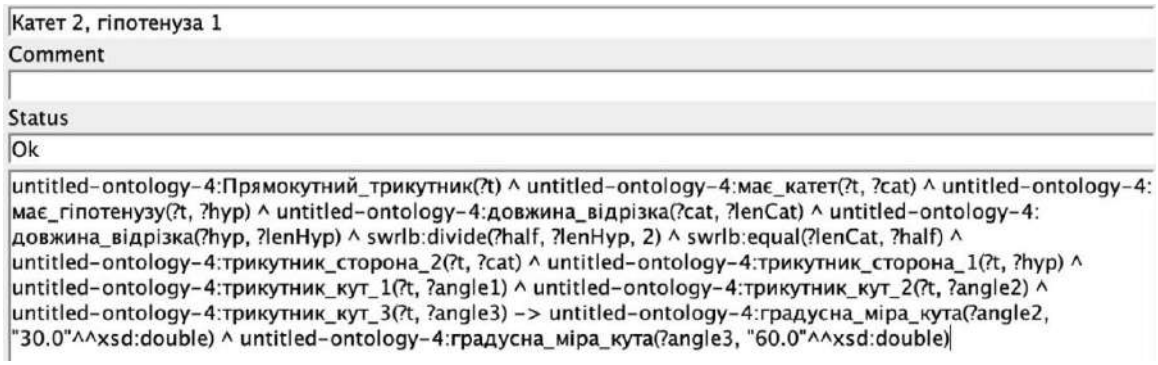


Рис 5.19. SWRL-правило. Катет 2. Гіпотенуза 1

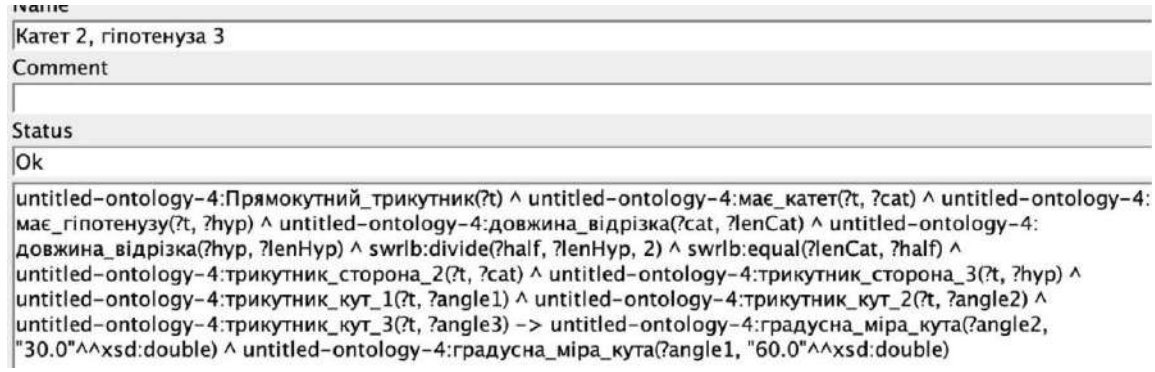


Рис 5.20. SWRL-правило. Катет 2. Гіпотенуза 3

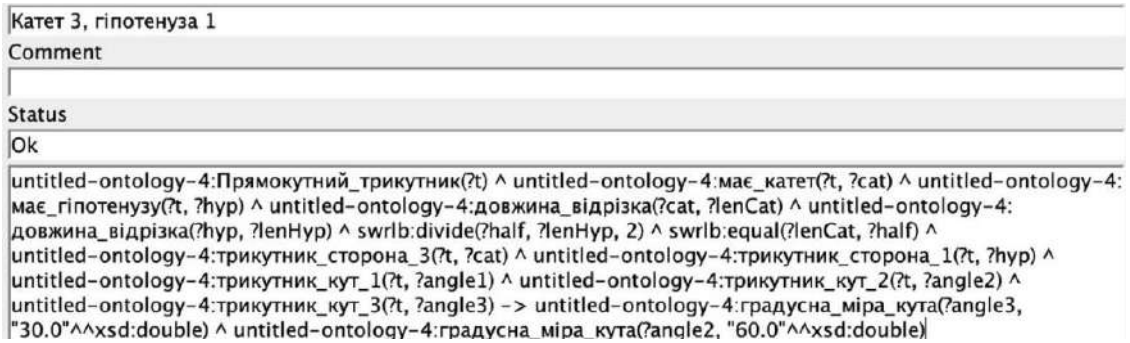


Рис 5.21. SWRL-правило. Катет 3. Гіпотенуза 1

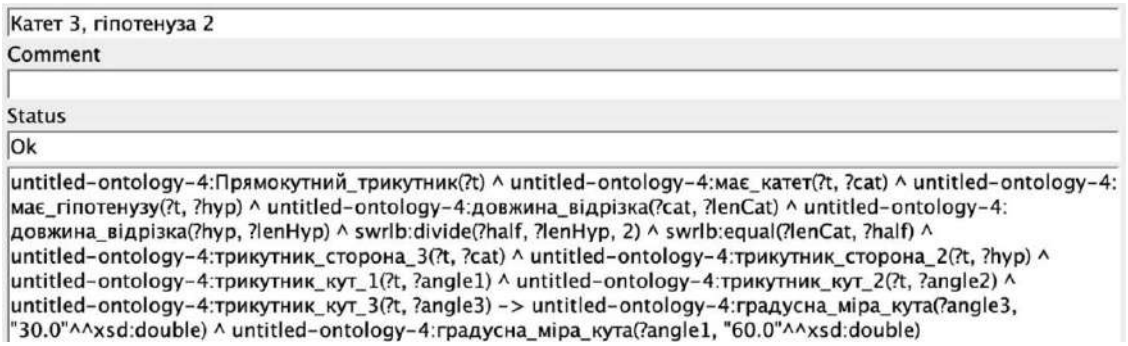


Рис 5.22. SWRL-правило. Катет 3. Гіпотенуза 2

Наступні правила визначають, що якщо два кути мають однакову градусну міру, а також збігаються за сторонами, то градусна міра одного з них передається іншому:

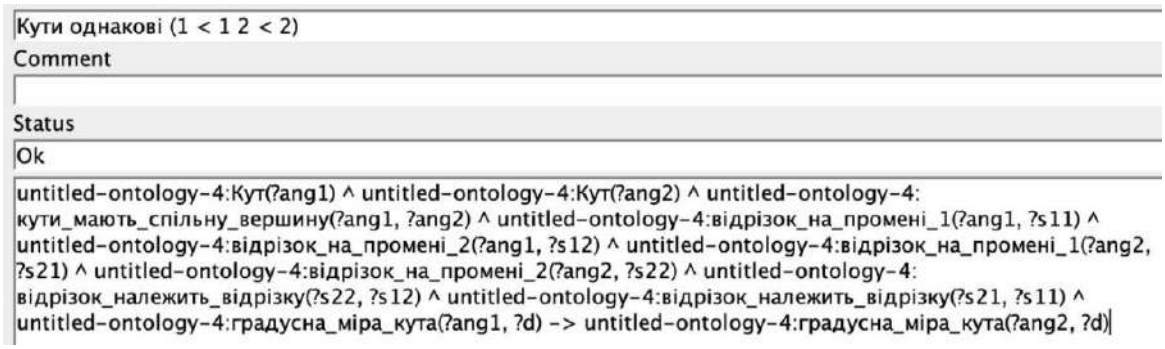


Рис 5.23. SWRL-правило. Порівняння кутів 1

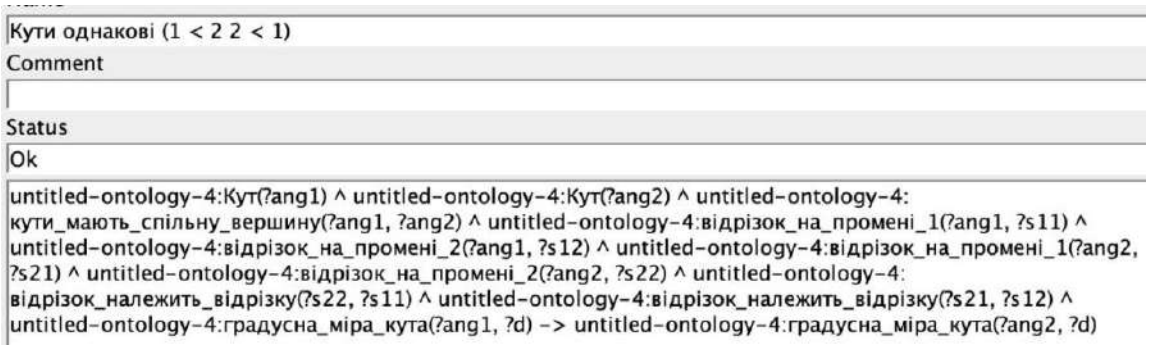


Рис 5.24. SWRL-правило. Порівняння кутів 2

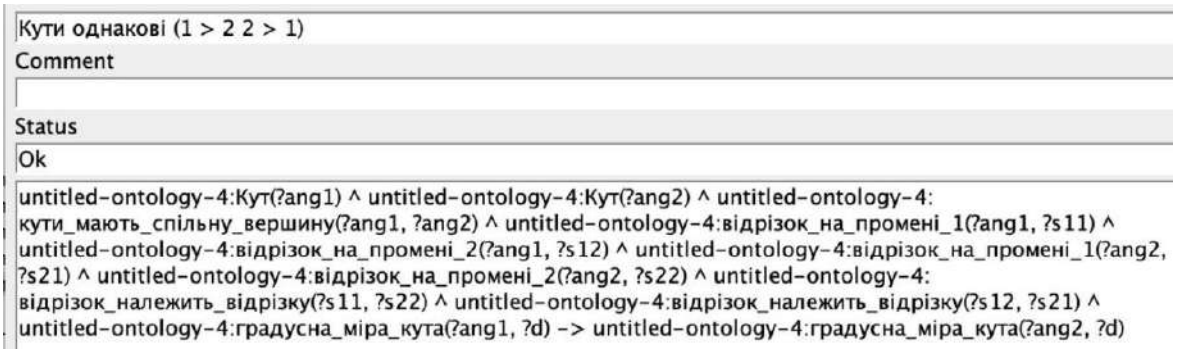


Рис 5.25. SWRL-правило. Порівняння кутів 3

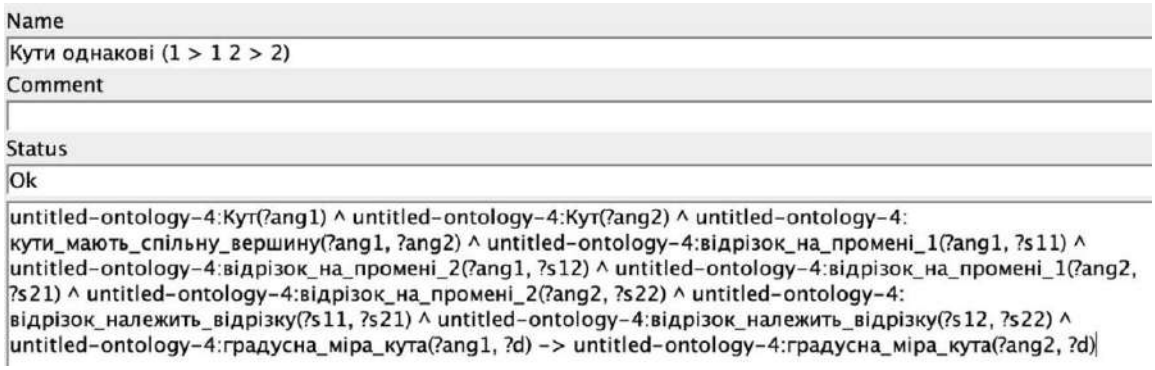


Рис 5.26. SWRL-правило. Порівняння кутів 4

Наступне правило виводить градусну міру суміжного кута:

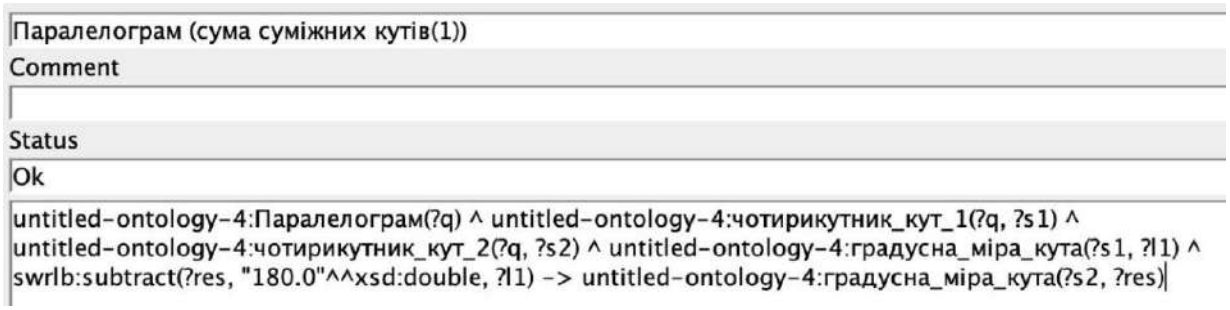


Рис 5.27. SWRL-правило. Сума суміжних кутів

Далі розглянемо міркування різонера:

Рис 5.28. Результат розрахунку відношення різонером

Рис 5.29. Результат розрахунку кута різонером

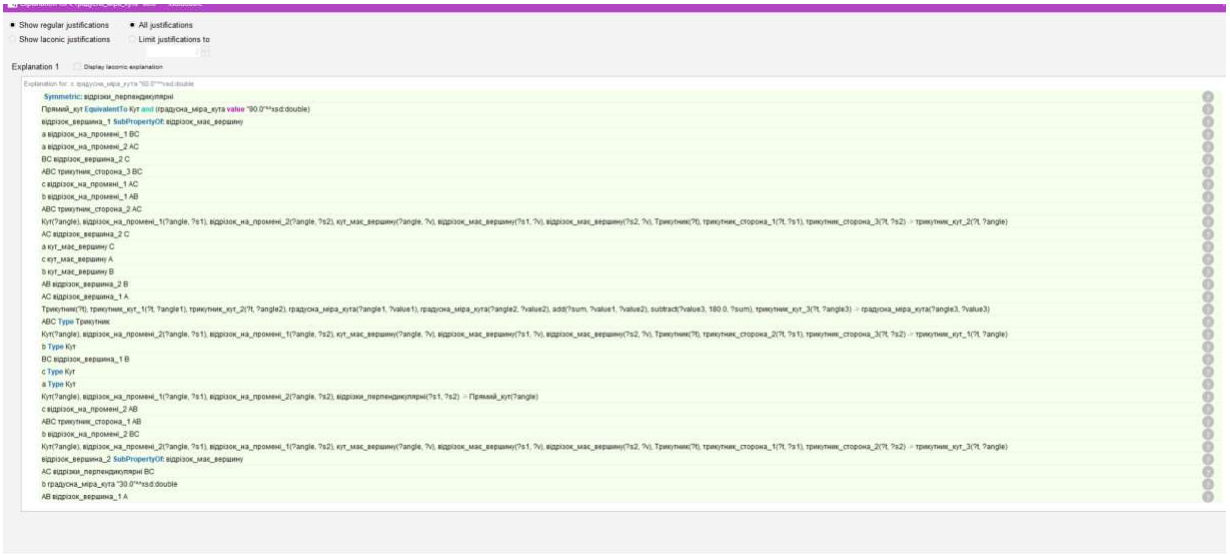


Рис 5.30. Аргументація різонера

Рис 5.31. Результати роботи різонера

Після отримання результатів та кроків логічного висновування, користувач може почати розв’язок задачі. Графічний інтерфейс відображає можливі кроки, та очікує введення відповідей. Якщо результат поточного етапу правильний, то він відобразиться як виконаний, інакше буде повідомлено про помилку.

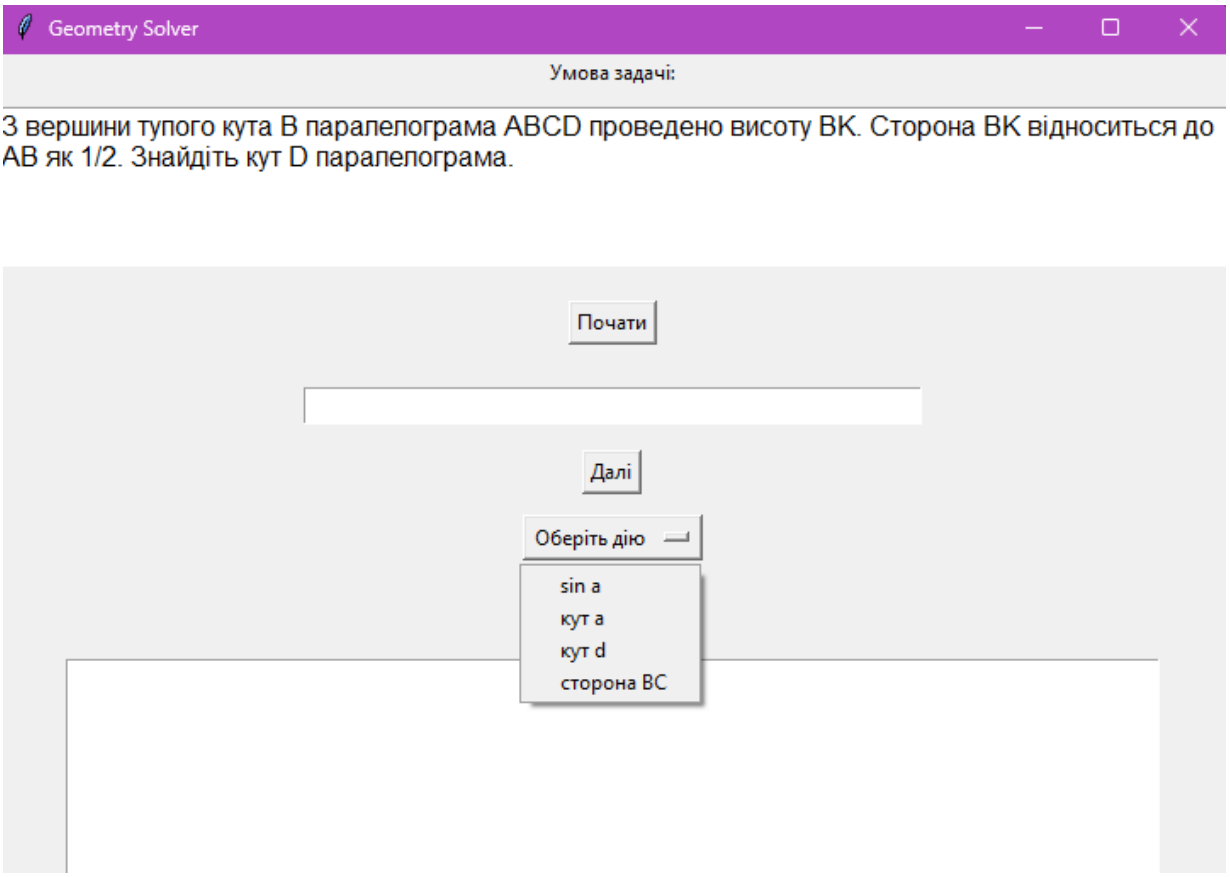


Рис 5.32. Графічний інтерфейс. Вибір дії

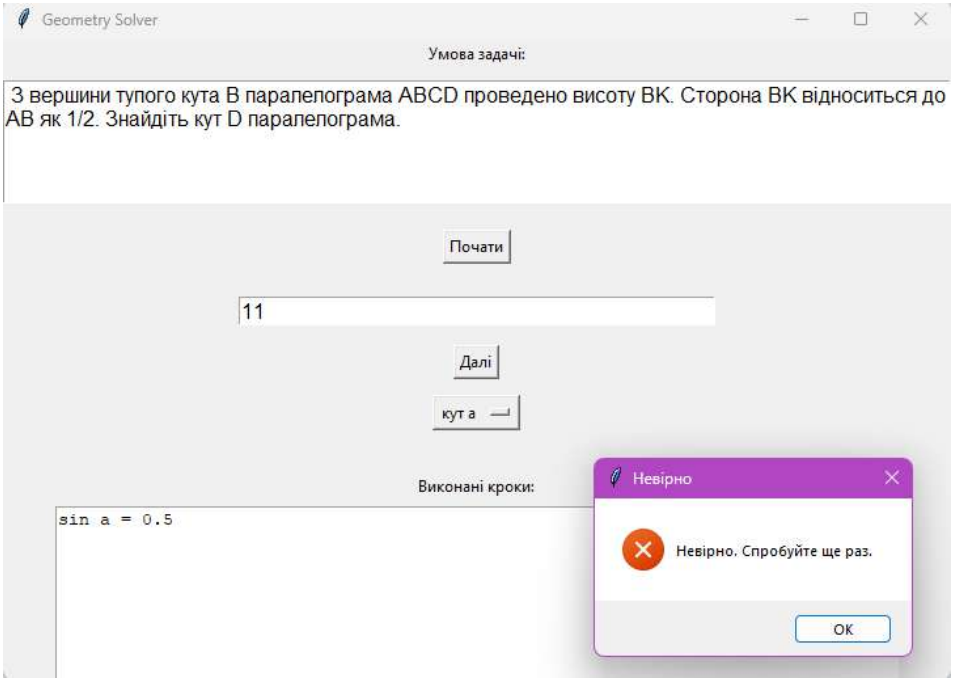


Рис 5.33. Графічний інтерфейс. Повідомлення про помилку

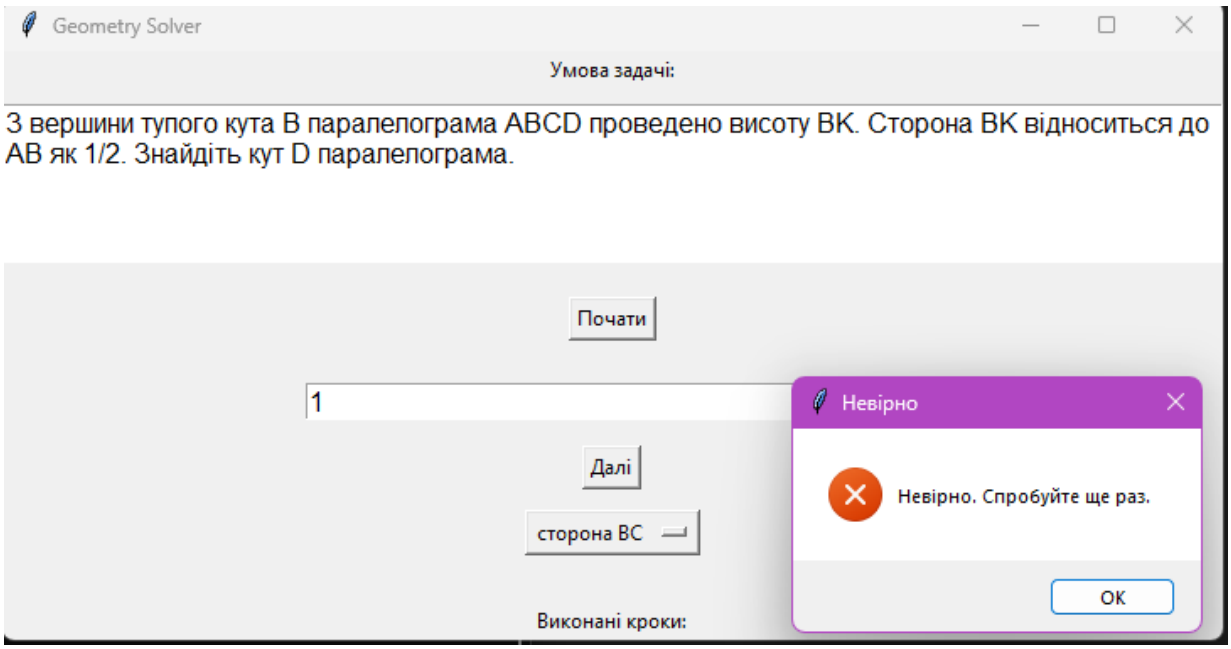


Рис 5.34. Відображення невдало обраного кроку в інтерфейсі

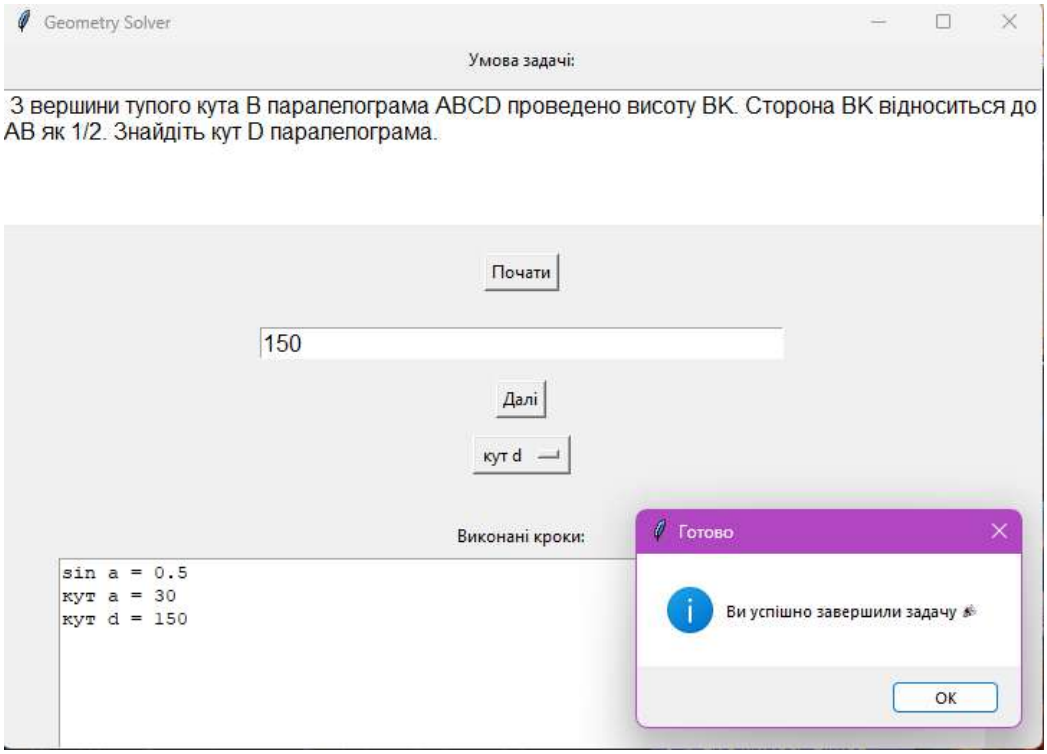


Рис 5.35. Завершення виконання завдання

Умова 2: Прямокутний трикутник має довжину катетів 3 та 8 см. Знайдіть площу вписаного кола.

Мовний аналіз задачі повертає нам таку структуру індивідів:

```
# newdoc
# newpar
# sent_id = 1
# text = Прямокутний трикутник має довжину катетів 6 та 8 см.
1       Прямокутний     Прямокутний     ADJ     Ao-msnf Case=Nom|Gender=Masc|Number=Sing        2       amod    _       _
2       трикутник       трикутник       NOUN    Ncmsnn  Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing   3       nsubj   _
_
3       має     мати    VERB    Vmpip3s Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin        0       r
oot     _       _
4       довжину довжина ADJ     Ao-fsas Case=Acc|Gender=Fem|Number=Sing 5       amod    _       _
5       катетів катет   NOUN    Ncmpay  Animacy=Anim|Case=Acc|Gender=Masc|Number=Plur   3       obj     _       _
6       6       6       NUM     Mlc-a   Case=Acc|NumType=Card|Uninflect=Yes     9       nummod:gov      _       _
7       та      та      CCONJ   Ccs     _       8       cc      _       _
8       8       8       NUM     Mlc-a   Case=Acc|NumType=Card|Uninflect=Yes     6       compound        _       _
9       см      см      NOUN    Y       Abbr=Yes|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur|Uninflect=Yes    5       f
lat:title       _       SpaceAfter=No
10      .       .       PUNCT   U       _       3       punct   _       _

# sent_id = 2
# text = Знайдіть радіус вписаного та описаного кола
1       Знайдіть        Знайдити        VERB    Vmem-2p Aspect=Perf|Mood=Imp|Number=Plur|Person=2|VerbForm=Fin  0       r
oot     _       _
2       радіус  радіус  NOUN    Ncmsan  Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing   1       obj     _       _
3       вписаного       вписаний        ADJ     Ao-nsgf Case=Gen|Gender=Neut|Number=Sing        6       amod    _       _
4       та      та      CCONJ   Ccs     _       5       cc      _       _
5       описаного       описаний        ADJ     Ao-nsgf Case=Gen|Gender=Neut|Number=Sing        3       conj    _       _
6       кола    коло    NOUN    Ncnsgn  Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing   2       nmod    _       SpacesAf
ter=\n
```

Рис 5.36. Результати мовного аналізу

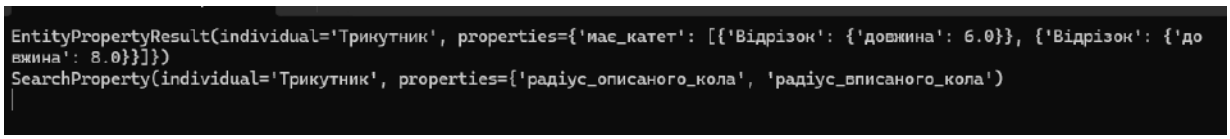


Рис 5.37. Виявлені індивіди

Основні SWRL правила, що необхідні для розв’язання даної задачі:

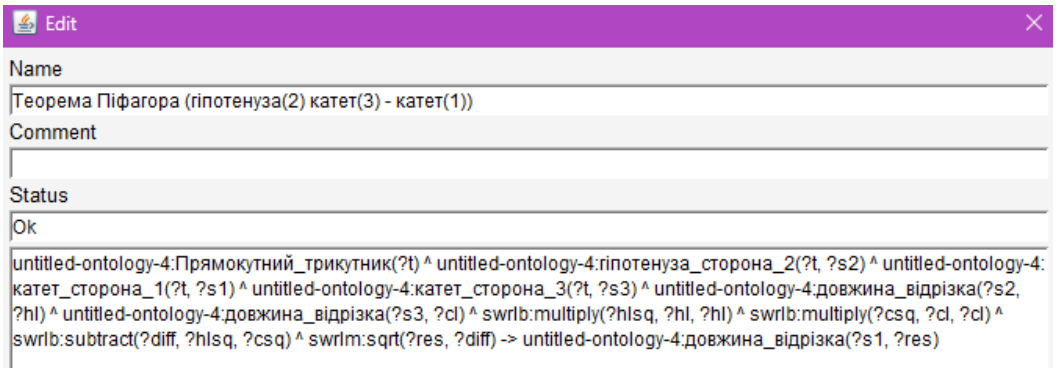


Рис 5.38. Обрахунок гіпотенузи за теоремою Піфагора

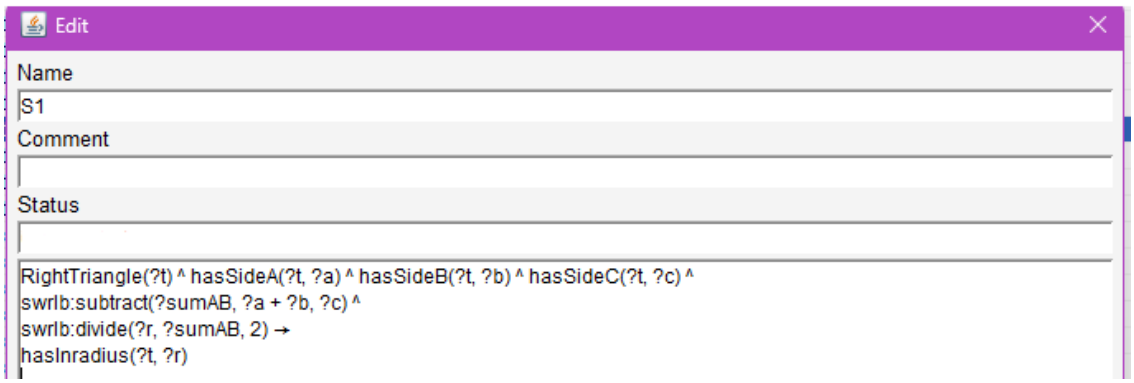


Рис 5.39. Обрахунок внутрішнього радіуса

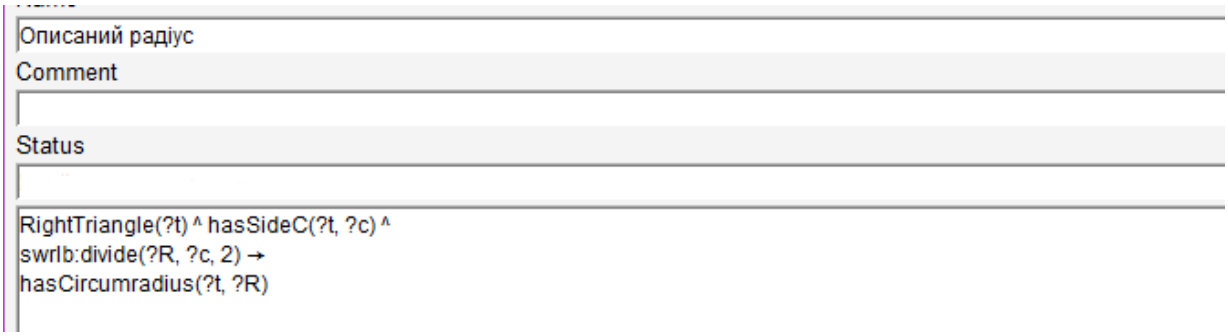


Рис 5.40. Обрахунок радіуса описаного кола

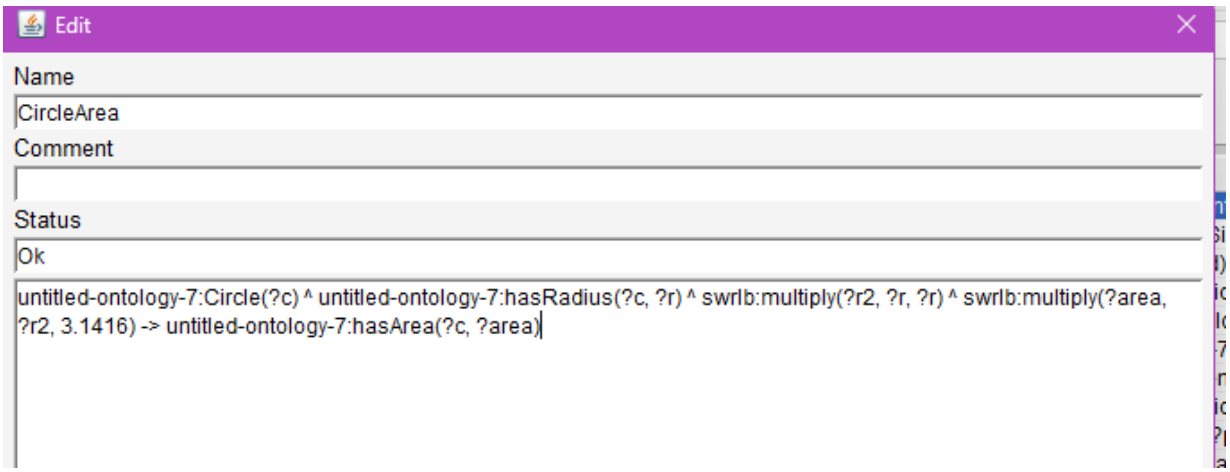


Рис 5.41. Обрахунок площі вписаного кола

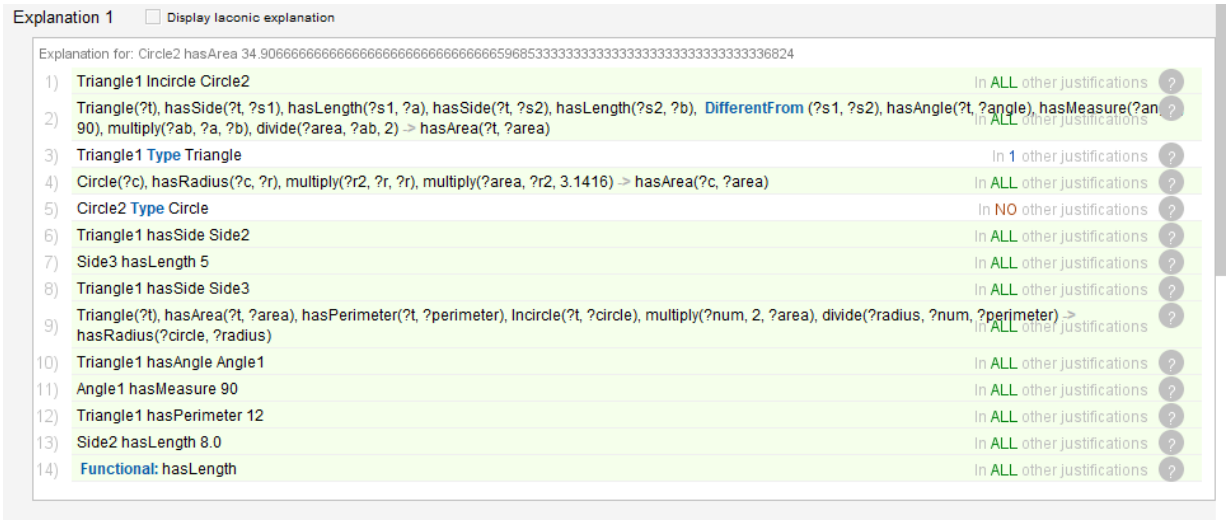


Рис 5.42. Результат висновування площі вписаного кола

Далі розглянемо застосування цієї задачі в графічному інтерфейсі:

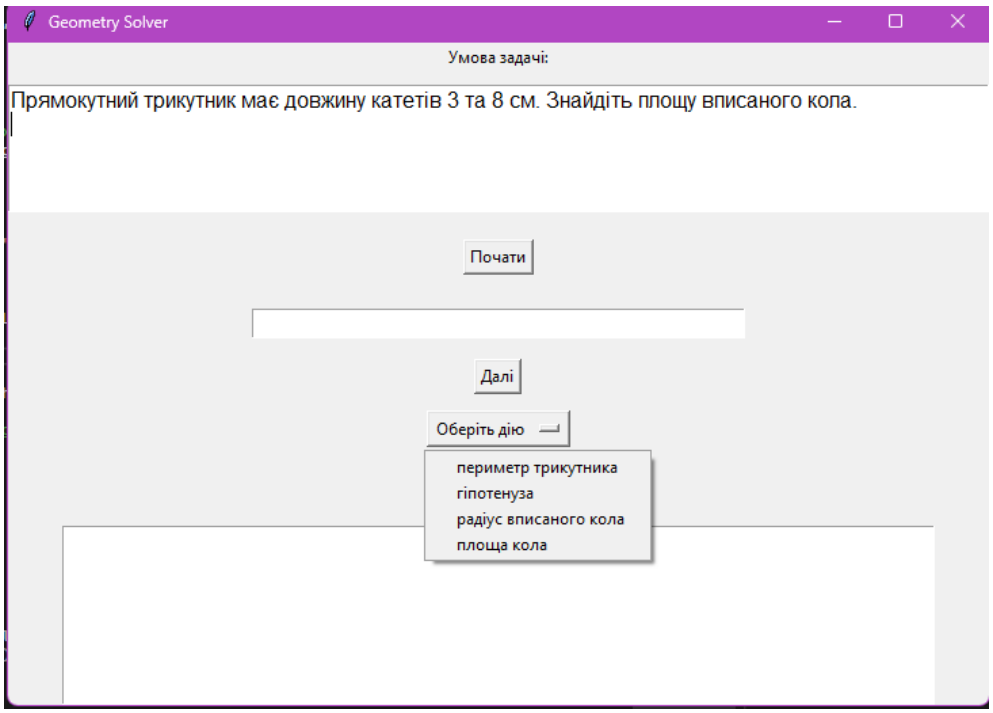


Рис 5.43. Вибір наступного кроку розв’язку

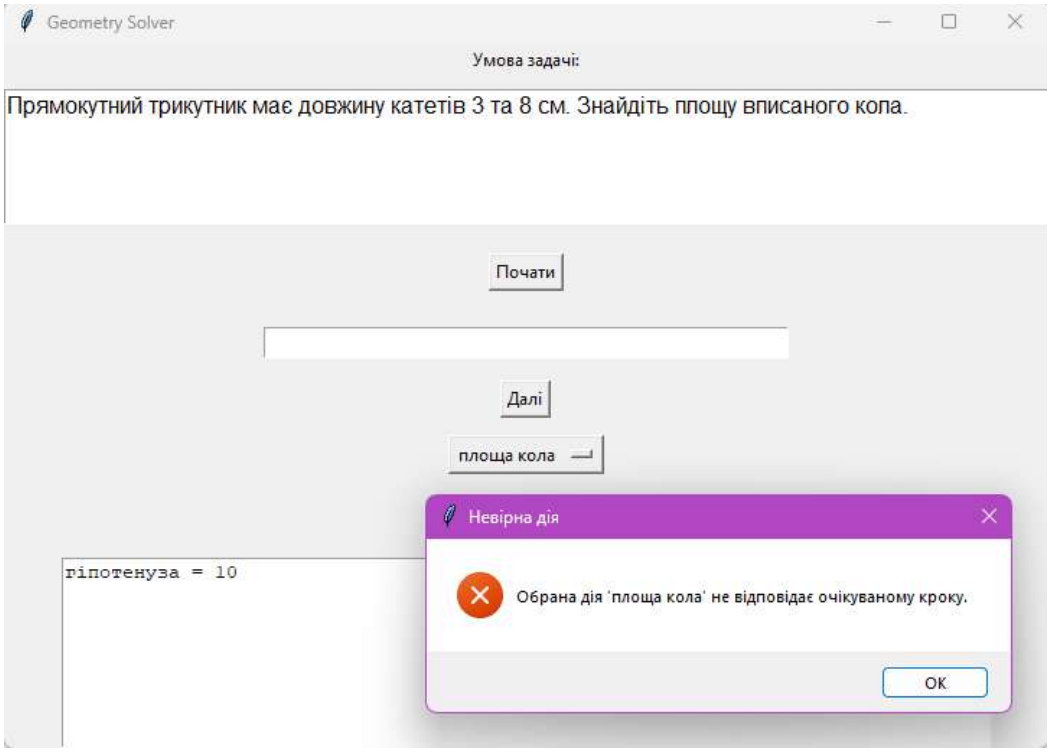


Рис 5.44. Помилковий вибір кроку

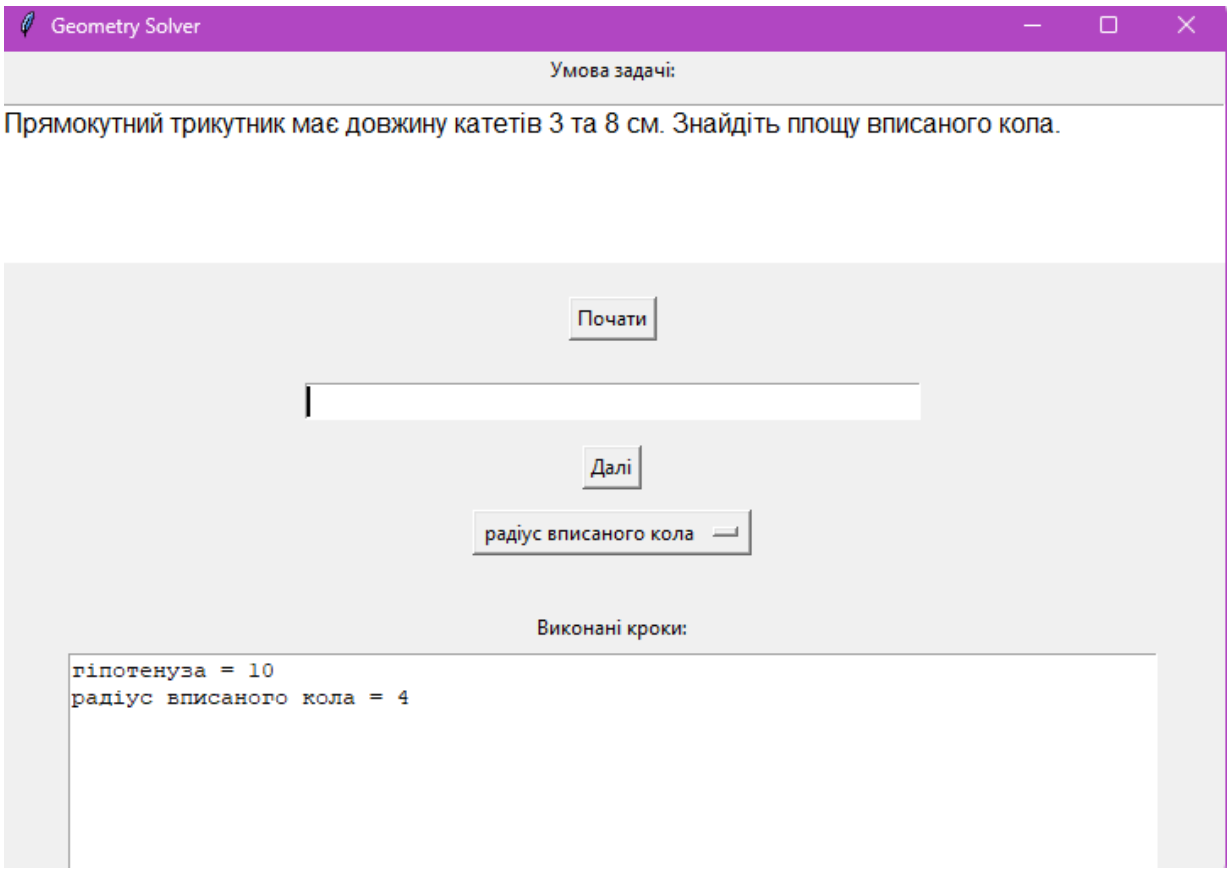


Рис 5.45. Помилкова відповідь етапу

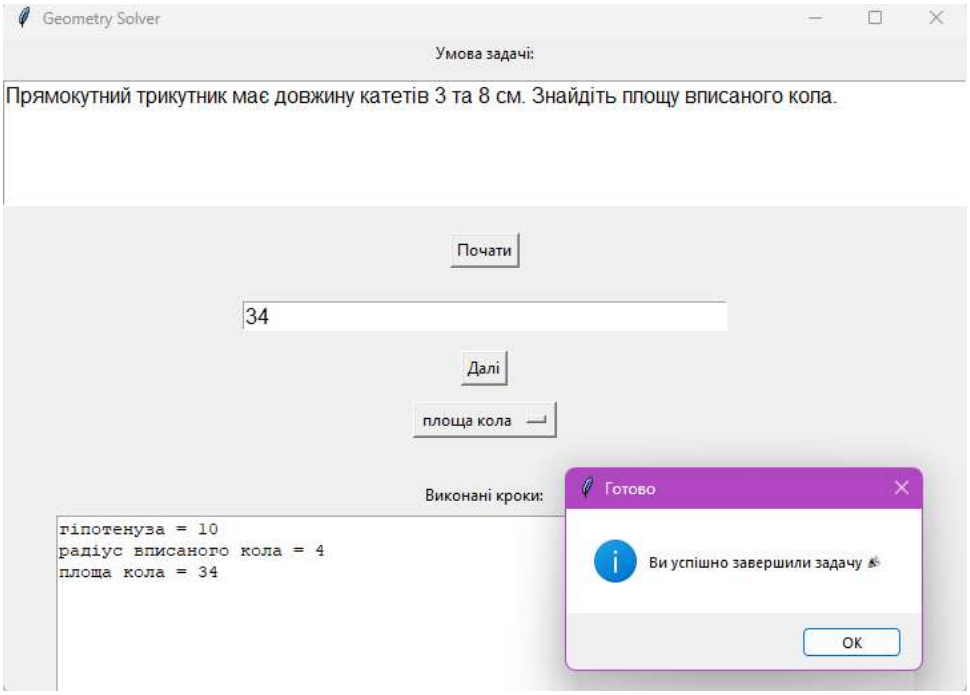
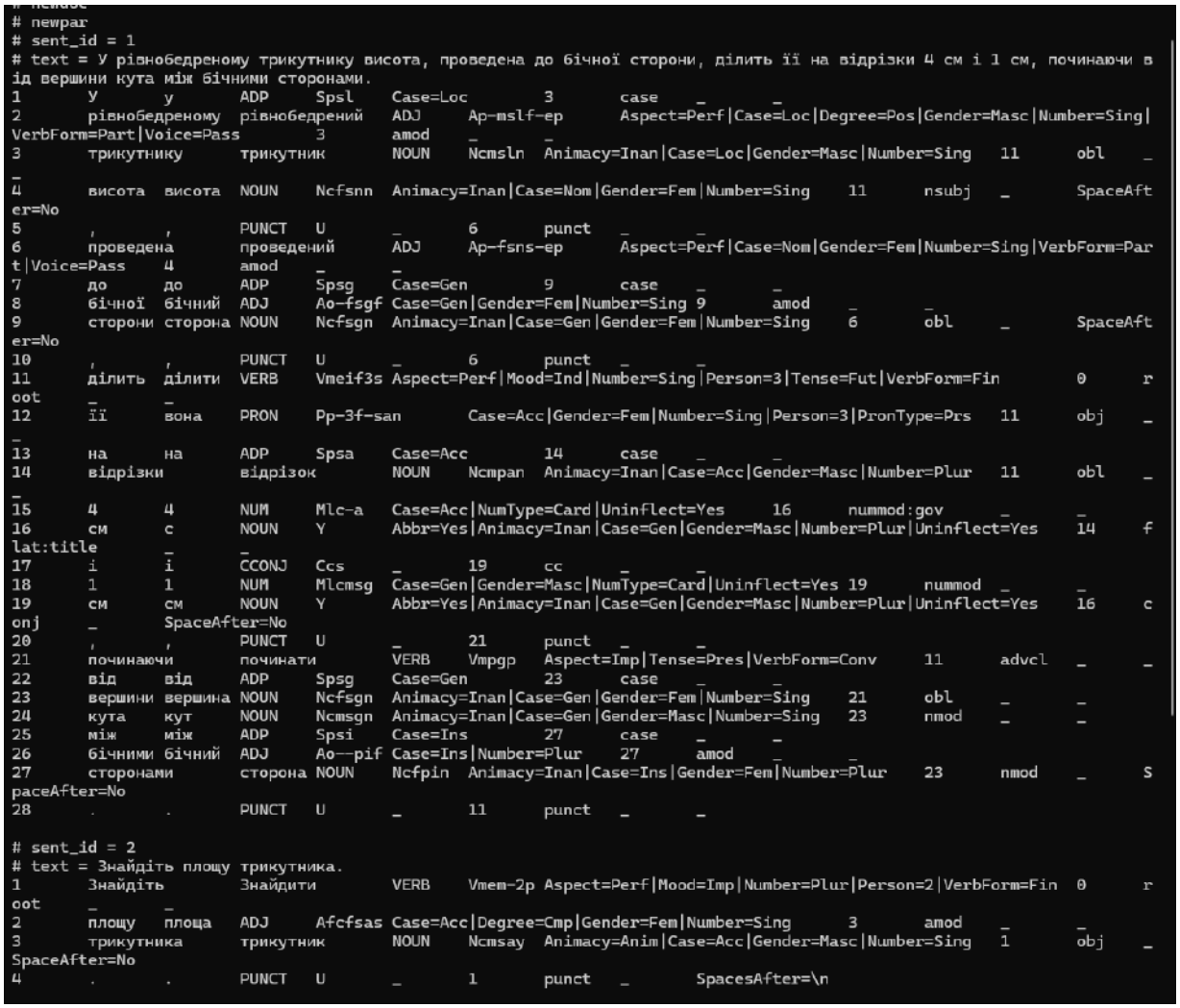


Рис 5.46. Успішне розв’язання задачі

Умова 3: У рівнобедреному трикутнику висота, проведена до бічної сторони, ділить її на відрізки 4 см і 1 см, починаючи від вершини кута між бічними сторонами. Знайдіть площу трикутника.

Аналіз умови:

- Маємо рівнобедрений трикутник, отже дві сторони рівні, а також кут між ними (назвемо його А).
- З точки вершини (назвемо її А) проведена висота АН, яка падає на бічну сторону ВС, не на основу.
- Ця висота ділить сторону ВС на відрізки 4 см і 1 см, тобто ВН = 4 см, НС = 1 см.
- Потрібно знайти площу трикутника АВ



```
# newdoc
# newpar
# sent_id = 1
# text = У рівнобедреному трикутнику висота, проведена до бічної сторони, ділить її на відрізки 4 см і 1 см, починаючи в
ід вершини кута між бічними сторонами.
1       У       у       ADP     Spsl    Case=Loc        3       case    _       _
2       рівнобедреному  рівнобедрений   ADJ     Ap-mslf-ep      Aspect=Perf|Case=Loc|Degree=Pos|Gender=Masc|Number=Sing|
VerbForm=Part|Voice=Pass        3       amod    _       _
3       трикутнику      трикутник       NOUN    Ncmsln  Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing   11      obl     _
_
4       висота  висота  NOUN    Ncfsnn  Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing    11      nsubj   _       SpaceAft
er=No
5       ,       ,       PUNCT   U       _       6       punct   _       _
6       проведена       проведений      ADJ     Ap-fsns-ep      Aspect=Perf|Case=Nom|Gender=Fem|Number=Sing|VerbForm=Par
t|Voice=Pass    4       amod    _       _
7       до      до      ADP     Spsg    Case=Gen        9       case    _       _
8       бічної  бічний  ADJ     Ao-fsgf Case=Gen|Gender=Fem|Number=Sing 9       amod    _       _
9       сторони сторона NOUN    Ncfsgn  Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing    6       obl     _       SpaceAft
er=No
10      ,       ,       PUNCT   U       _       6       punct   _       _
11      ділить  ділити  VERB    Vmeif3s Aspect=Perf|Mood=Ind|Number=Sing|Person=3|Tense=Fut|VerbForm=Fin        0       r
oot     _       _
12      її      вона    PRON    Pp-3f-san       Case=Acc|Gender=Fem|Number=Sing|Person=3|PronType=Prs   11      obj     _
_
13      на      на      ADP     Spsa    Case=Acc        14      case    _       _
14      відрізки        відрізок        NOUN    Ncmpan  Animacy=Inan|Case=Acc|Gender=Masc|Number=Plur   11      obl     _
_
15      4       4       NUM     Mlc-a   Case=Acc|NumType=Card|Uninflect=Yes     16      nummod:gov      _       _
16      см      с       NOUN    Y       Abbr=Yes|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur|Uninflect=Yes    14      f
lat:title       _       _
17      і       і       CCONJ   Ccs     _       19      cc      _       _
18      1       1       NUM     Mlcmsg  Case=Gen|Gender=Masc|NumType=Card|Uninflect=Yes 19      nummod  _       _
19      см      см      NOUN    Y       Abbr=Yes|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur|Uninflect=Yes    16      c
onj     _       SpaceAfter=No
20      ,       ,       PUNCT   U       _       21      punct   _       _
21      починаючи       починати        VERB    Vmpgp   Aspect=Imp|Tense=Pres|VerbForm=Conv     11      advcl   _       _
22      від     від     ADP     Spsg    Case=Gen        23      case    _       _
23      вершини вершина NOUN    Ncfsgn  Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing    21      obl     _       _
24      кута    кут     NOUN    Ncmsgn  Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing   23      nmod    _       _
25      між     між     ADP     Spsi    Case=Ins        27      case    _       _
26      бічними бічний  ADJ     Ao--pif Case=Ins|Number=Plur    27      amod    _       _
27      сторонами       сторона NOUN    Ncfpin  Animacy=Inan|Case=Ins|Gender=Fem|Number=Plur    23      nmod    _       S
paceAfter=No
28      .       .       PUNCT   U       _       11      punct   _       _

# sent_id = 2
# text = Знайдіть площу трикутника.
1       Знайдіть        Знайдити        VERB    Vmem-2p Aspect=Perf|Mood=Imp|Number=Plur|Person=2|VerbForm=Fin  0       r
oot     _       _
2       площу   площа   ADJ     Afcfsas Case=Acc|Degree=Cmp|Gender=Fem|Number=Sing      3       amod    _       _
3       трикутника      трикутник       NOUN    Ncmsay  Animacy=Anim|Case=Acc|Gender=Masc|Number=Sing   1       obj     _
SpaceAfter=No
4       .       .       PUNCT   U       _       1       punct   _       SpacesAfter=\n
```

Рис 5.47. Результати мовного аналізу

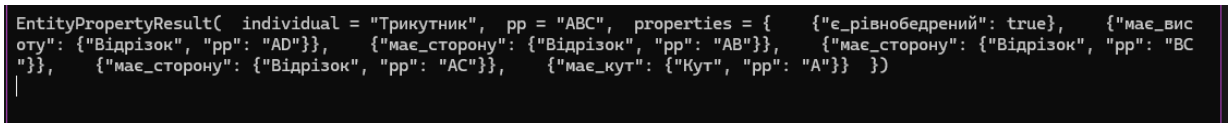


Рис 5.48. Результати визначення індивідів

Після мовного аналізу задачі, отримані індивіди завантажуються в онтологію та запускається процес висновування за SWRL правилами.

Наступні правила перебирають можливі бічні сторони трикутника та встановлюють, що у рівнобедреному трикутнику бічні сторони мають однакову довжину:

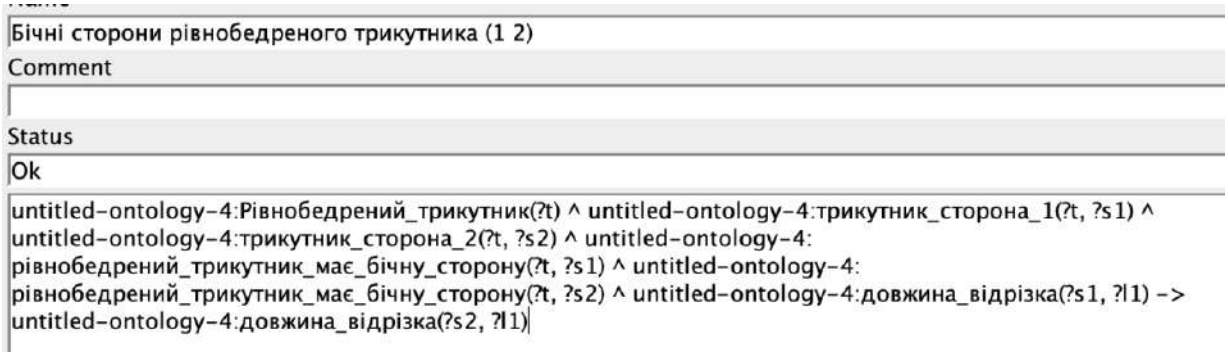


Рис 5.49. SWRL-правило. Бічні рівнобедреного сторони трикутника 1

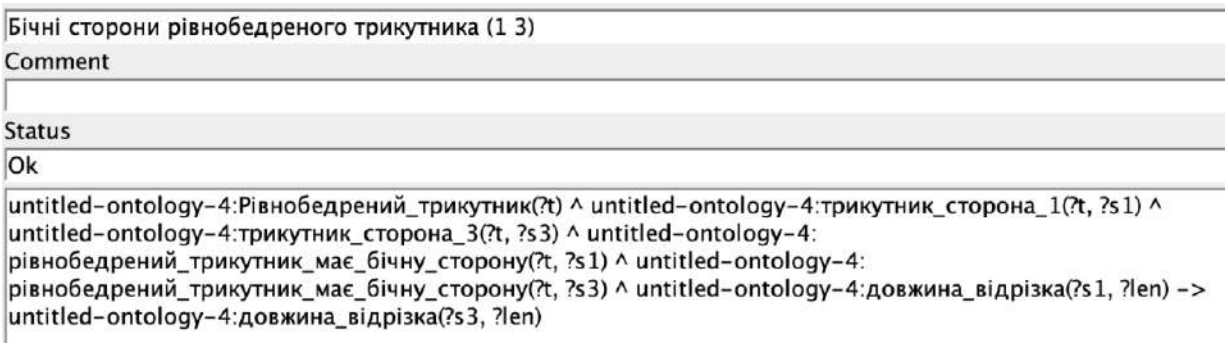


Рис 5.50. SWRL-правило. Бічні сторони рівнобедреного трикутника 2

Name

Бічні сторони рівнобедреного трикутника (2 1)

Comment

Status

Ok

```
untitled-ontology-4:Рівнобедрений_трикутник(?t) ^ untitled-ontology-4:трикутник_сторона_1(?t, ?s1) ^
untitled-ontology-4:трикутник_сторона_2(?t, ?s2) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s1) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s2) ^ untitled-ontology-4:довжина_відрізка(?s2, ?len) ->
untitled-ontology-4:довжина_відрізка(?s1, ?len)
```

Рис 5.51. SWRL-правило. Бічні сторони рівнобедреного трикутника 3

Бічні сторони рівнобедреного трикутника (2 3)

Comment

Status

Ok

```
untitled-ontology-4:Рівнобедрений_трикутник(?t) ^ untitled-ontology-4:трикутник_сторона_2(?t, ?s2) ^
untitled-ontology-4:трикутник_сторона_3(?t, ?s3) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s2) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s3) ^ untitled-ontology-4:довжина_відрізка(?s2, ?len) ->
untitled-ontology-4:довжина_відрізка(?s3, ?len)
```

Рис 5.52. SWRL-правило. Бічні сторони рівнобедреного трикутника 4

Бічні сторони рівнобедреного трикутника (3 1)

Comment

Status

Ok

```
untitled-ontology-4:Рівнобедрений_трикутник(?t) ^ untitled-ontology-4:трикутник_сторона_1(?t, ?s1) ^
untitled-ontology-4:трикутник_сторона_3(?t, ?s3) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s1) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s3) ^ untitled-ontology-4:довжина_відрізка(?s3, ?len) ->
untitled-ontology-4:довжина_відрізка(?s1, ?len)
```

Рис 5.53. SWRL-правило. Бічні сторони рівнобедреного трикутника 5

Name

Бічні сторони рівнобедреного трикутника (3 2)

Comment

Status

Ok

```
untitled-ontology-4:Рівнобедрений_трикутник(?t) ^ untitled-ontology-4:трикутник_сторона_2(?t, ?s2) ^
untitled-ontology-4:трикутник_сторона_3(?t, ?s3) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s2) ^ untitled-ontology-4:
рівнобедрений_трикутник_має_бічну_сторону(?t, ?s3) ^ untitled-ontology-4:довжина_відрізка(?s3, ?len) ->
untitled-ontology-4:довжина_відрізка(?s2, ?len)
```

Рис 5.54. SWRL-правило. Бічні сторони рівнобедреного трикутника 6

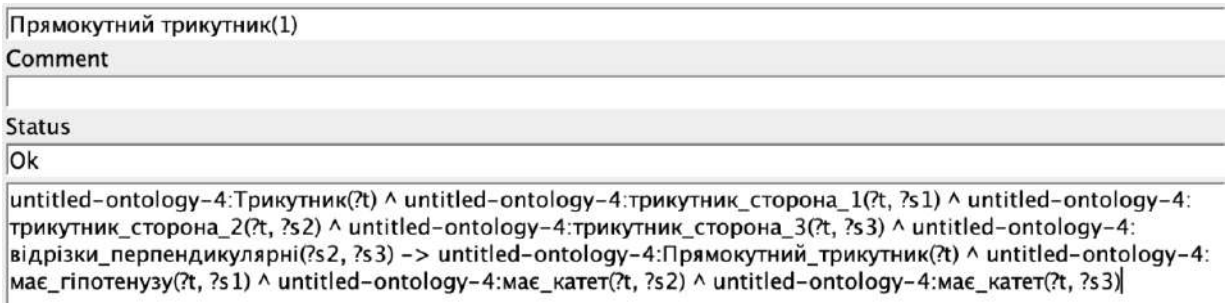


Рис 5.55. SWRL-правило. Прямокутний трикутник 1

Нижче правила перебирають сторони трикутника і визначають, що якщо дві сторони в трикутнику перпендикулярні, то цей трикутник прямокутний, у якому ці сторони є катетами, а третя сторона є гіпотенузою:

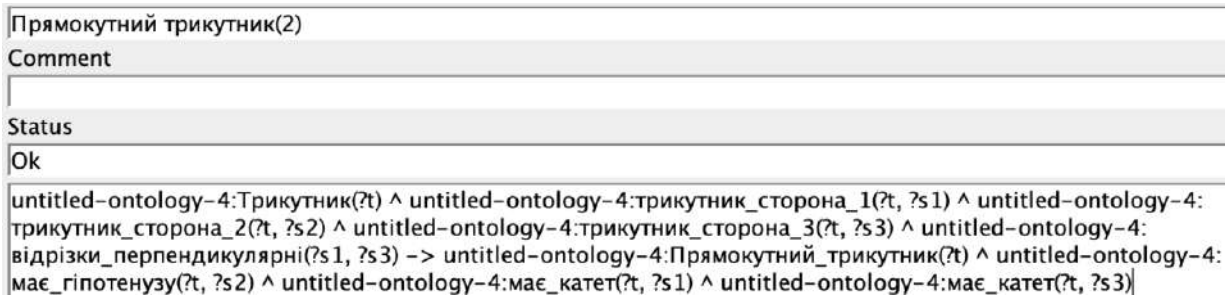


Рис 5.56. SWRL-правило. Прямокутний трикутник 2

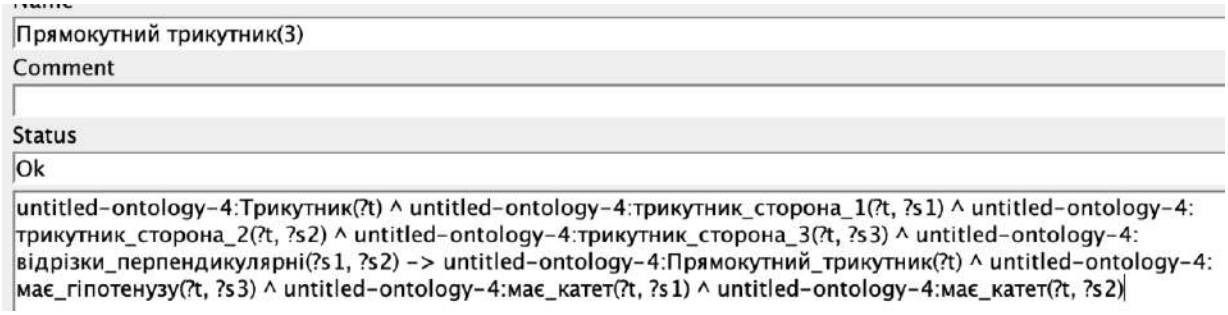


Рис 5.57. SWRL-правила. Прямокутний трикутник 3

Наступні правила перебирають усі можливі сторони прямокутного трикутника і обчислюють довжину третьої сторони, якщо відомі дві інші:

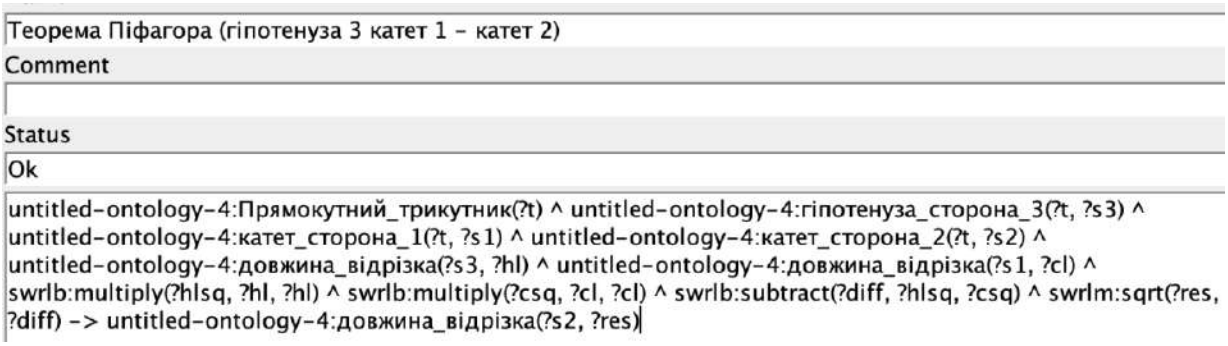


Рис 5.58. SWRL-правила. Теорема Піфагора 1

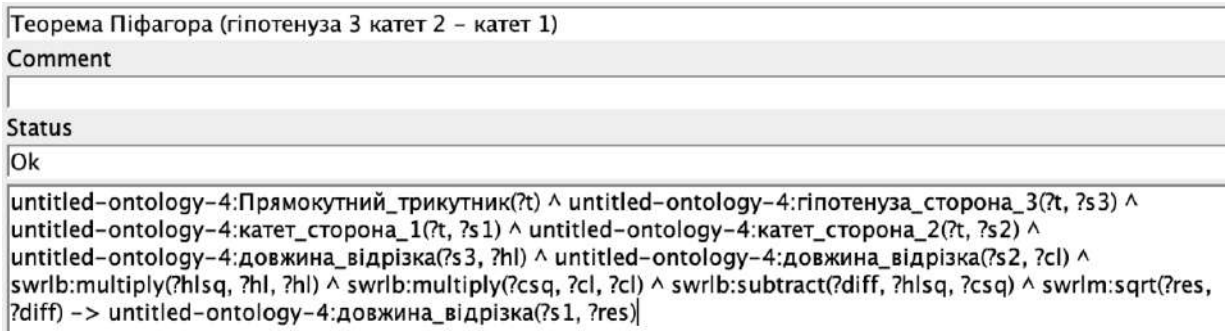


Рис 5.59. SWRL-правила. Теорема Піфагора 2

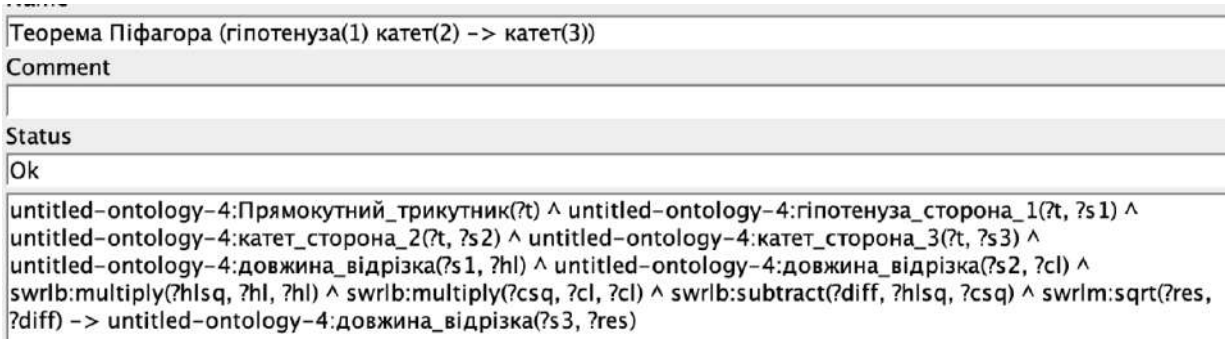


Рис 5.60. SWRL-правила. Теорема Піфагора 3

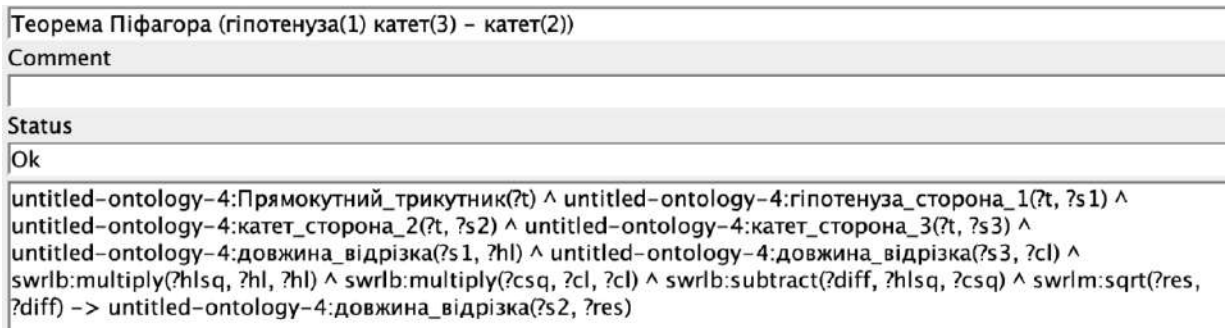


Рис 5.61. SWRL-правила. Теорема Піфагора 4

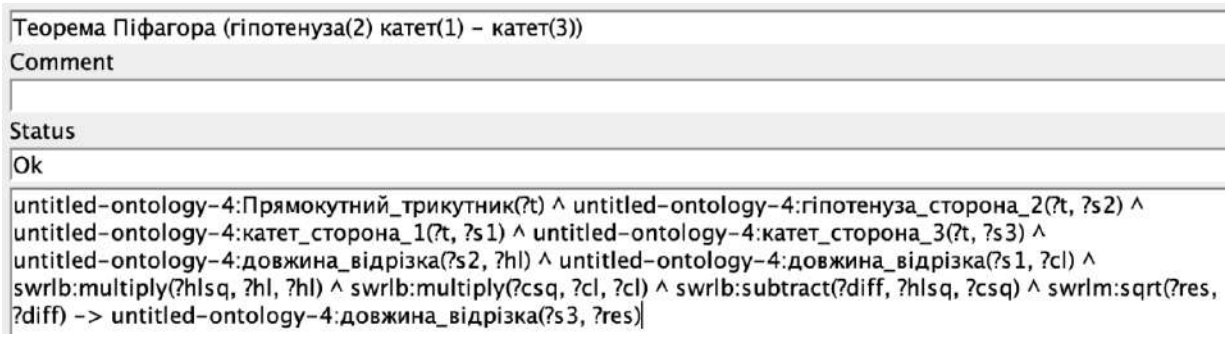


Рис 5.62. SWRL-правила. Теорема Піфагора 5

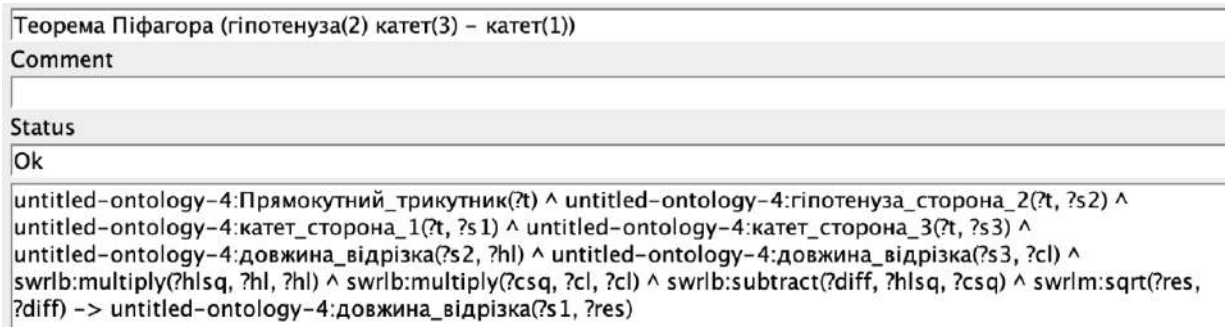


Рис 5.63. SWRL-правила. Теорема Піфагора 6

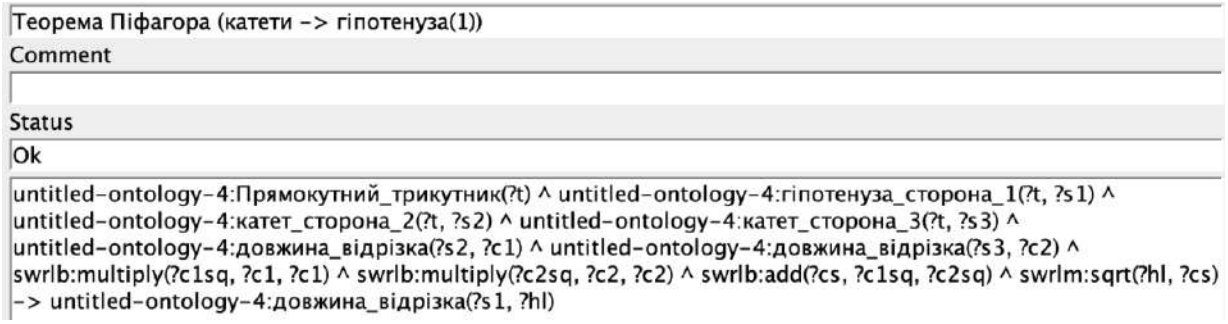


Рис 5.64. SWRL-правила. Теорема Піфагора 7

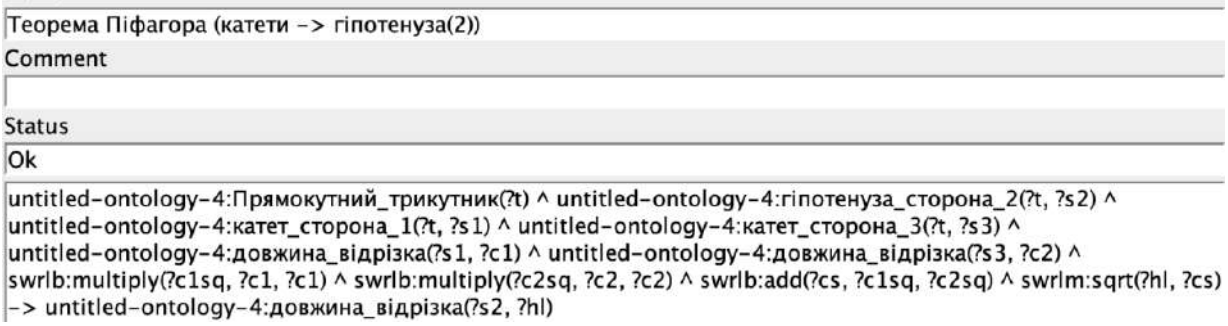


Рис 5.65. SWRL-правила. Теорема Піфагора 8

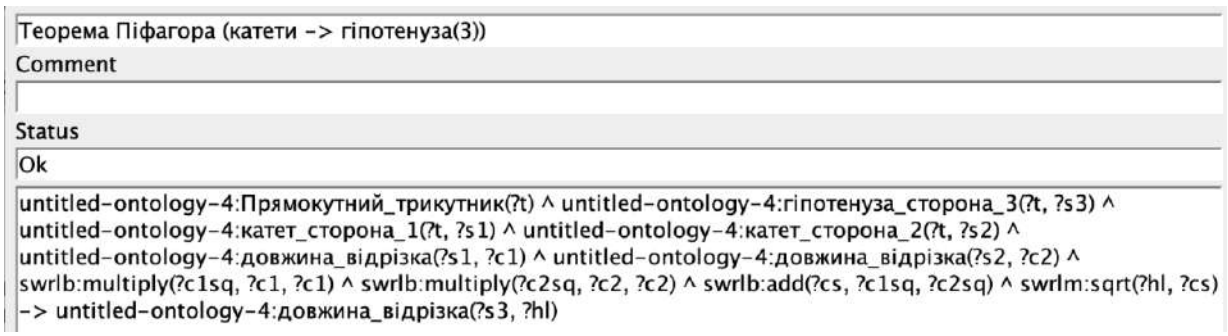


Рис 5.66. SWRL-правила. Теорема Піфагора 9

Далі правила перебирають спочатку перебирають вершини та сторони і обраховують площу трикутника, якщо відома довжина висоти та відповідної сторони, до якої проведена ця висота:

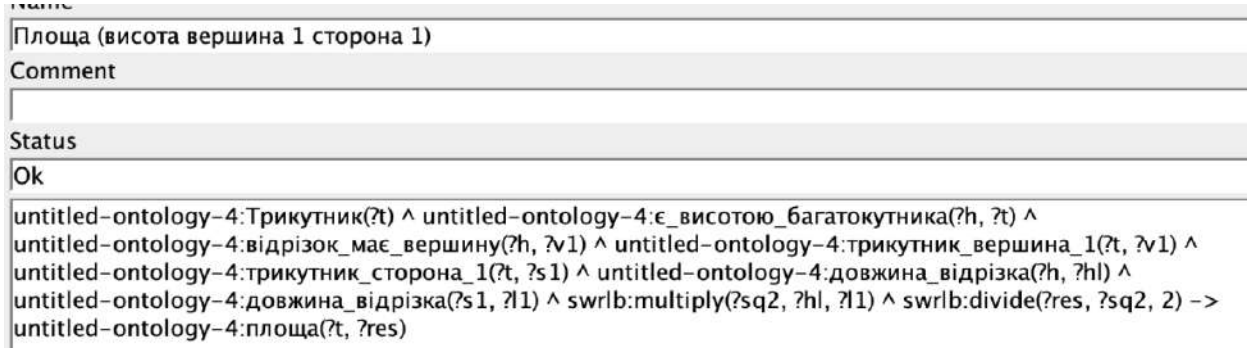


Рис 5.67. SWRL-правила. Площа трикутника 1

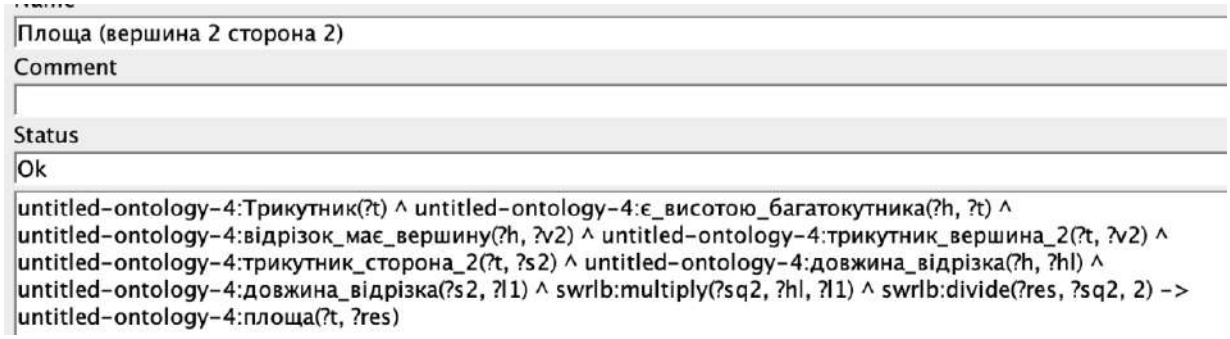


Рис 5.68. SWRL-правила. Площа трикутника 2

Площа (вершина 3 сторона 3)

Comment

Status

Ok

untitled-ontology-4:Трикутник(?t) ^ untitled-ontology-4:є_висотою_багатокутника(?h, ?t) ^
untitled-ontology-4:відрізок_має_вершину(?h, ?v3) ^ untitled-ontology-4:трикутник_вершина_3(?t, ?v3) ^
untitled-ontology-4:трикутник_сторона_2(?t, ?s3) ^ untitled-ontology-4:довжина_відрізка(?h, ?hl) ^
untitled-ontology-4:довжина_відрізка(?s3, ?l1) ^ swrlb:multiply(?sq2, ?hl, ?l1) ^ swrlb:divide(?res, ?sq2, 2) ->
untitled-ontology-4:площа(?t, ?res)

Рис 5.69. SWRL-правила. Площа трикутника 3

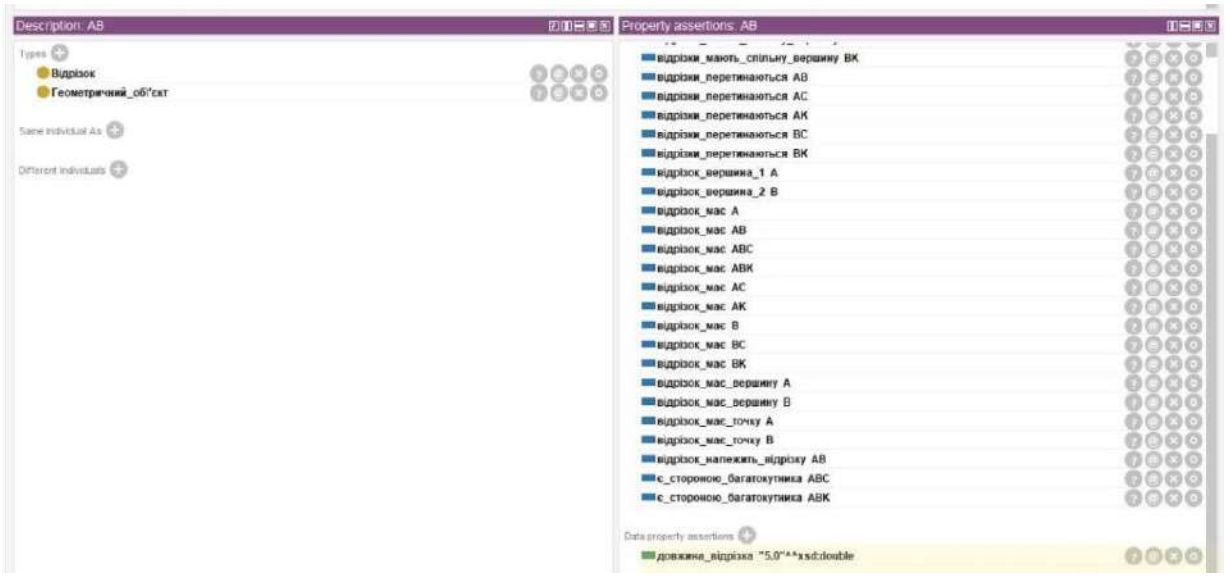


Рис 5.70. Результати висновування відрізка

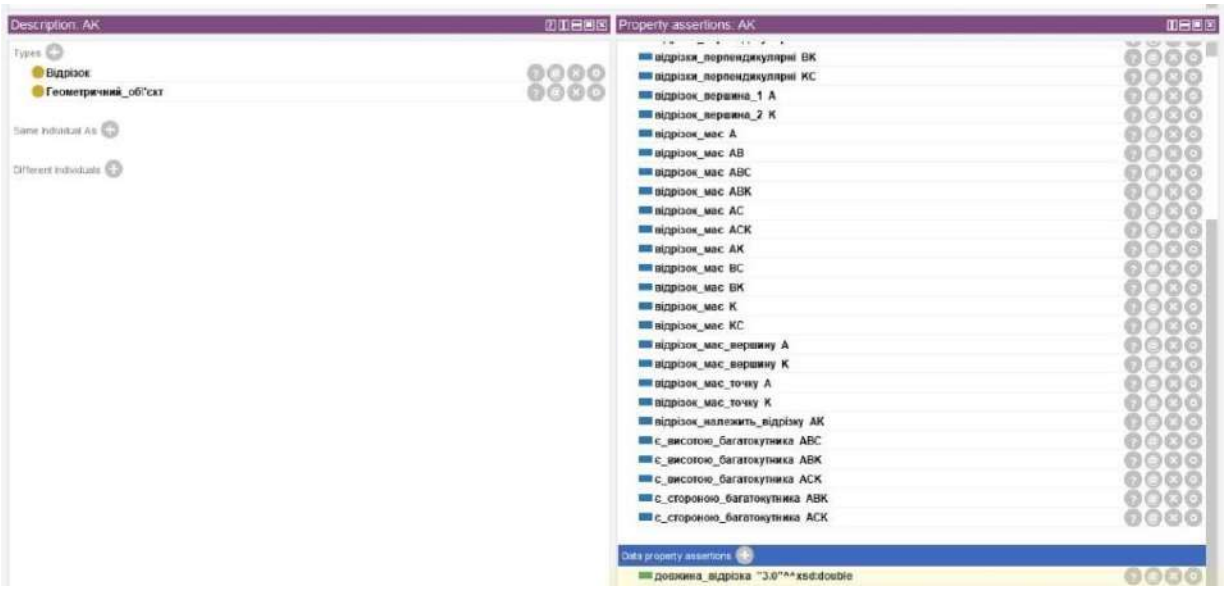


Рис 5.71. Результати висновування відрізка

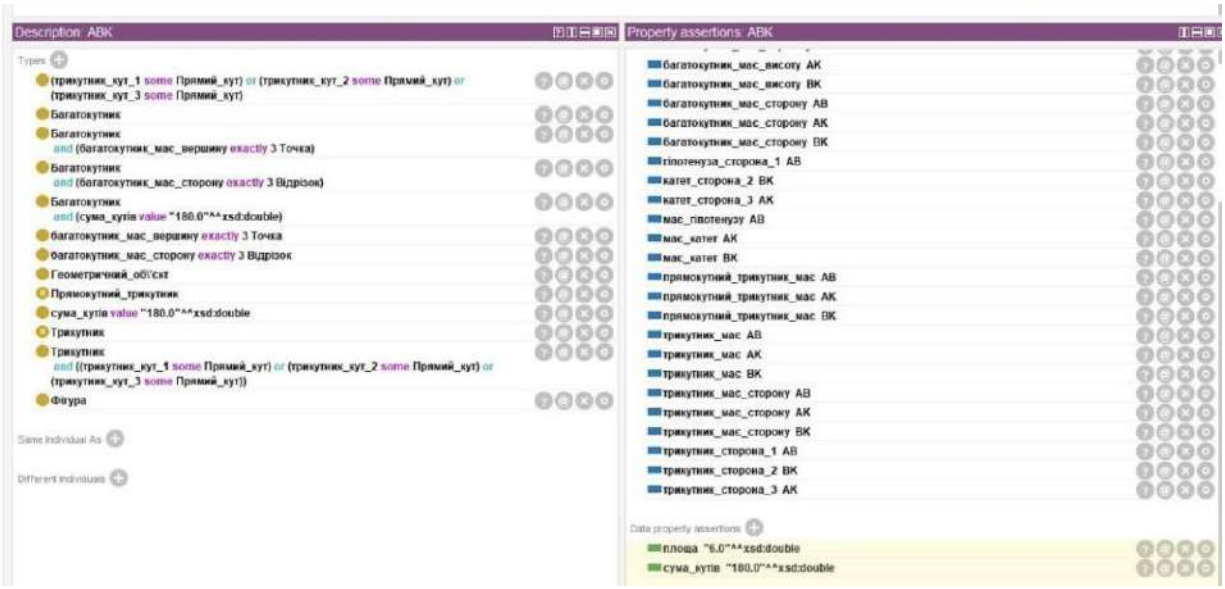


Рис 5.72. Результати висновування площі

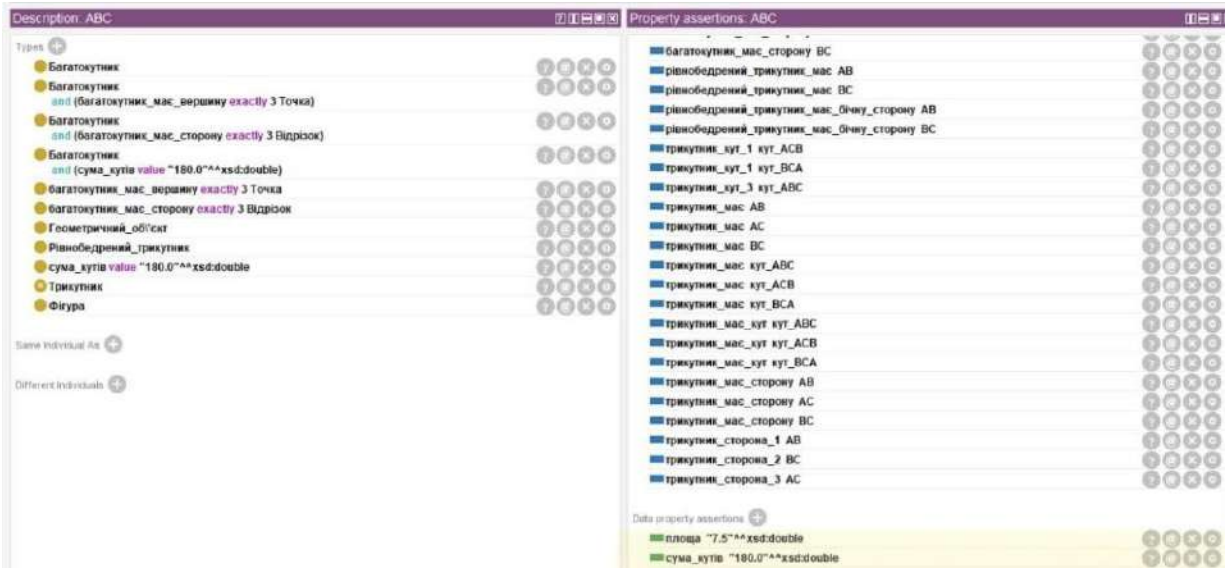


Рис 5.73. Фінальні результати висновування площі

Наступним кроком розглянемо алгоритм розв’язку задачі в графічному інтерфейсі користувача:

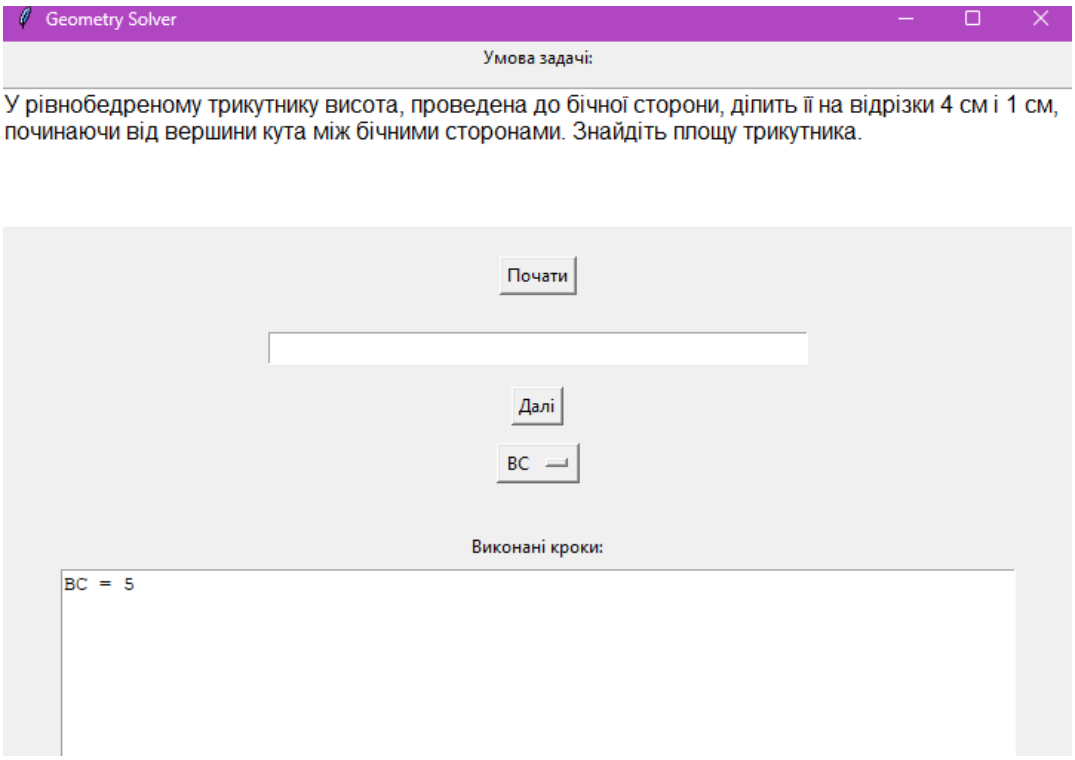


Рис 5.74. Інтерфейс користувача. Умова задачі

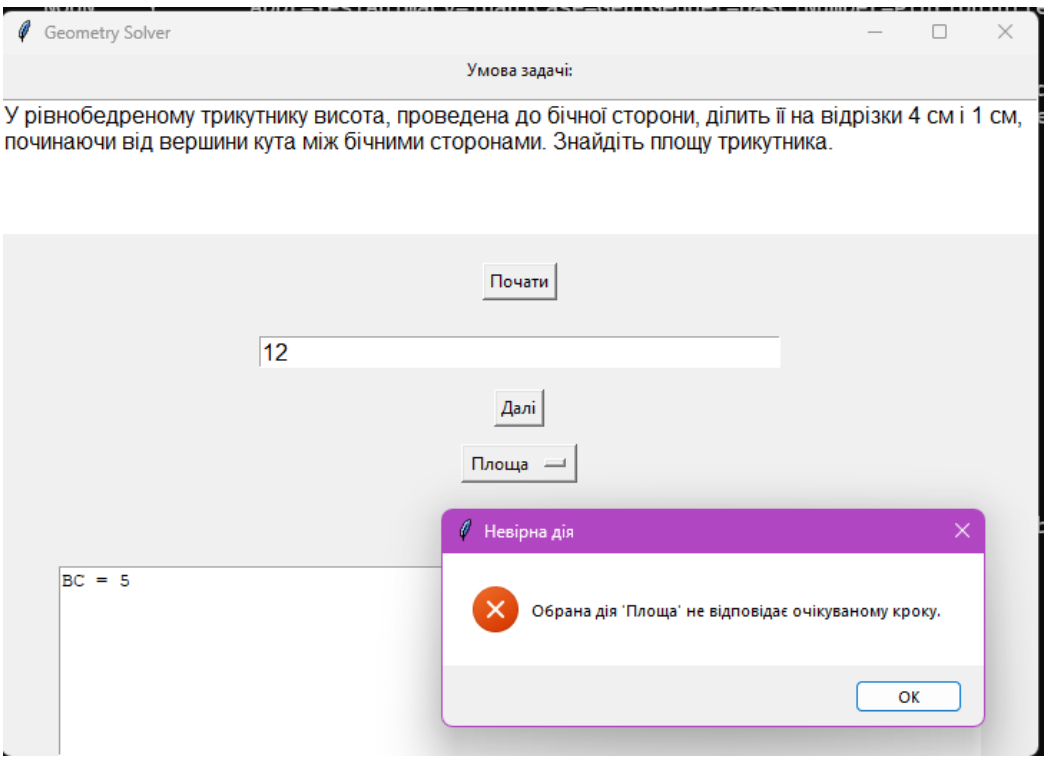


Рис 5.75. Інтерфейс користувача. Неправильно обрана дія

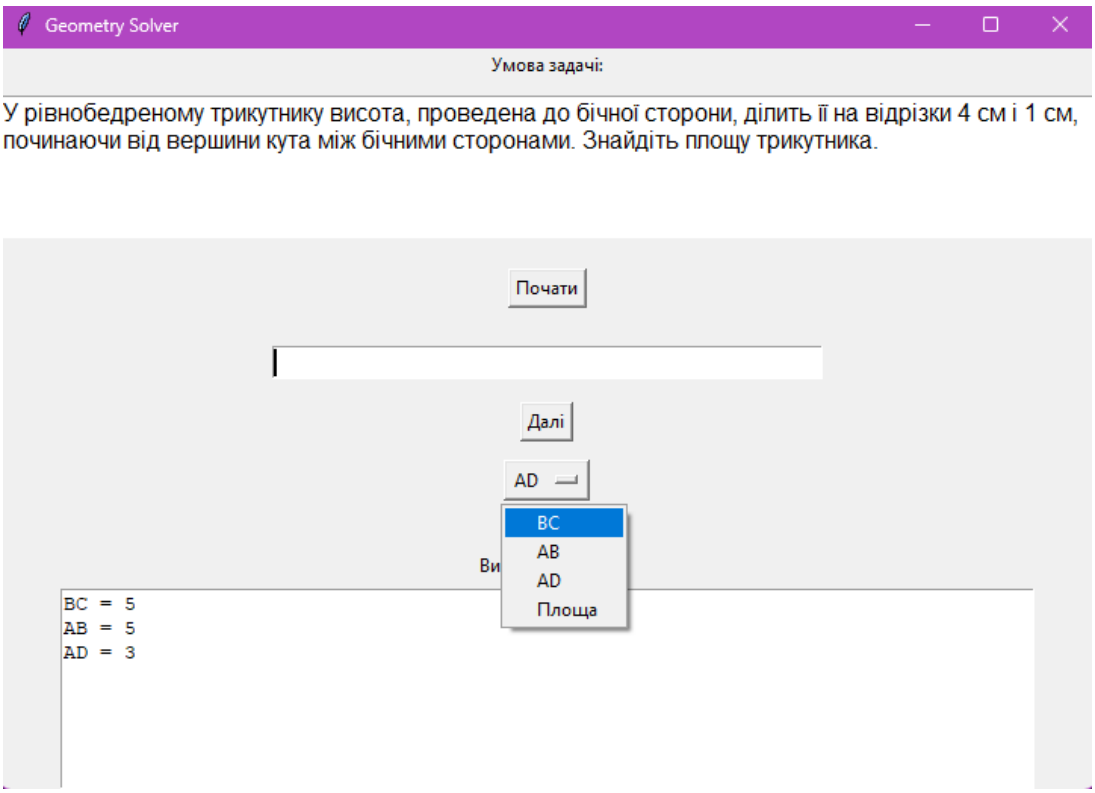


Рис 5.76. Інтерфейс користувача. Вибір кроку

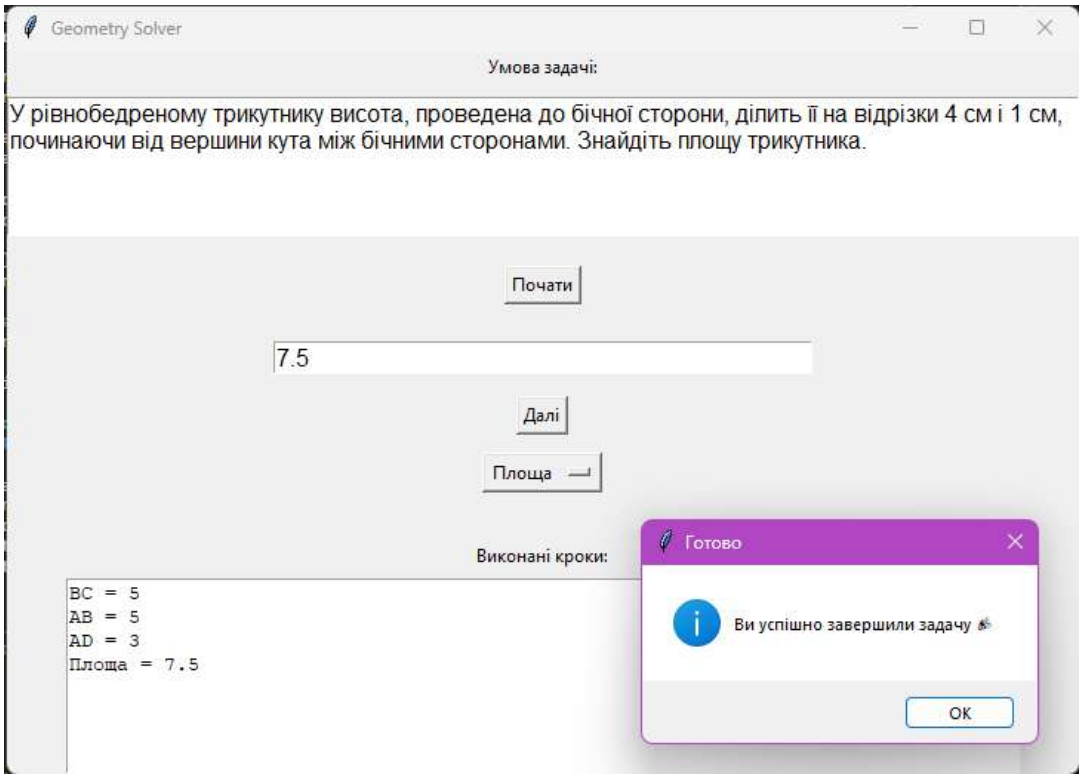


Рис 5.77. Інтерфейс користувача. Виконання задачі

Висновок

Отже, в рамках роботи було реалізовано рекомендаційну систему для вирішення геометричних задач з відображенням та перевіркою їх кроків. В ході розробки програмного рішення, було досліджено можливості лематизації та морфологічного аналізу природного тексту та змодельовано онтологічну базу знань в середовищі Protege, що відповідає необхідній предметній області.

Отримані результати показують загальну ефективність реалізації в розв’язуванні геометричних задач. Обробка природної мови дозволяє коректно структурувати вхідні дані та сформулювати ієрархію властивостей вказаних індивідів. В свою чергу онтологічна модель з використанням SWRL правил надає можливість отримувати повні покрокові рішення введених завдань.

Отримана система може стати потужним інструментом серед інших навчальних інтелектуальних систем й мати беззаперечне значення в покращенні та автоматизації освітнього процесу. Внутрішня структуризація правил, аксіом та знань з предметної області гарантує коректність та незмінність результатів розв’язків на відміну від рішень на основі штучного інтелекту.

Тому, поєднання систем на основі аналізу природної мови та онтологічних моделей є перспективним рішенням для розвитку інтелектуальних навчальних систем та забезпечує можливість здобувачам освіти отримувати алгоритм розв’язку, а не лише самі відповіді.

Список літератури

1. Худік Б. О. Застосування рекомендаційної системи в освітній сфері / Б. О. Худік // Державний університет інформаційно-комунікаційних технологій. - Київ, 2023. - URL: https://con.dut.edu.ua/index.php/communication/article/view/2721.
2. A free, open-source ontology editor and framework for building intelligent system [Електронний ресурс] // Stanford Center for Biomedical Informatics Research. - Режим доступу: https://protege.stanford.edu/.
3. Web Ontology Language (OWL) [Електронний ресурс] // W3C. - 2012. - Режим доступу: https://www.w3.org/OWL/.
4. SWRL: A Semantic Web Rule Language Combining OWL and RuleML [Електронний ресурс] // W3C. - 2004. - Режим доступу: https://www.w3.org/Submission/SWRL/.
5. Жежерун О. П., Смиш О. Р., Прудніков А. О. Підходи до побудови виводу в онтологічній базі знань / О. П. Жежерун, О. Р. Смиш, А. О. Прудніков. - Київ : Києво-Могилянська академія, 2023. - 7 с. - Режим доступу: https://doi.org/10.18523/2617-3808.2023.6.17-23.
6. Мерзляк А. Г. Геометрія : підручник для 7 кл. закладів загальної середньої освіти. - 2-ге вид. - Харків : Гімназія, 2020. - 240 с.
7. Gruber T. R. A translation approach to portable ontology specifications / T. R. Gruber // Knowledge Acquisition. - 1993. - Vol. 5, No. 2. - P. 199-220.
8. Noy N. F., McGuinness D. L. Ontology Development 101: A Guide to Creating Your First Ontology [Електронний ресурс] / N. F. Noy, D. L. McGuinness. - 2001. - Режим доступу: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.
9. Ricci F., Rokach L., Shapira B. Introduction to Recommender Systems Handbook // In: Ricci F., Rokach L., Shapira B., Kantor P. B. (eds) Recommender Systems Handbook. - Springer, 2011. - Режим доступу:

https://www.researchgate.net/publication/227268858_Recommender_Systems_Handbook.

10. Zeldes A. Universal Dependencies for Ukrainian: Morphosyntactic and Syntactic Annotation. - 2017.
11. Straka M., Hajic J., Straková J. UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing // Proceedings of LREC. - UFAL, 2016. - URL: https://ufal.mff.cuni.cz/udpipe.
12. Nivre J. та ін. Universal Dependencies v1: A Multilingual Treebank Collection // Proceedings of LREC 2016. - 2016. - Режим доступу: https://universaldependencies.org/.
13. Straka M., Hajic J., Straková J. UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing // *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. - European Language Resources Association (ELRA), 2016. - Режим доступу: https://aclanthology.org/L16-1680/.
14. Wan Li, Huaai Kang, Dongbo Ma, Weiwei Wei. SWRL Parallel Reasoning Implementation with Spark SQL // *IOP Conference Series: Materials Science and Engineering*. - 2020. - DOI: 10.1088/1757-899X/719/1/012020.
15. Horridge M., Bechhofer S. The OWL API: A Java API for OWL Ontologies // School of Computer Science, University of Manchester.

Додаток А

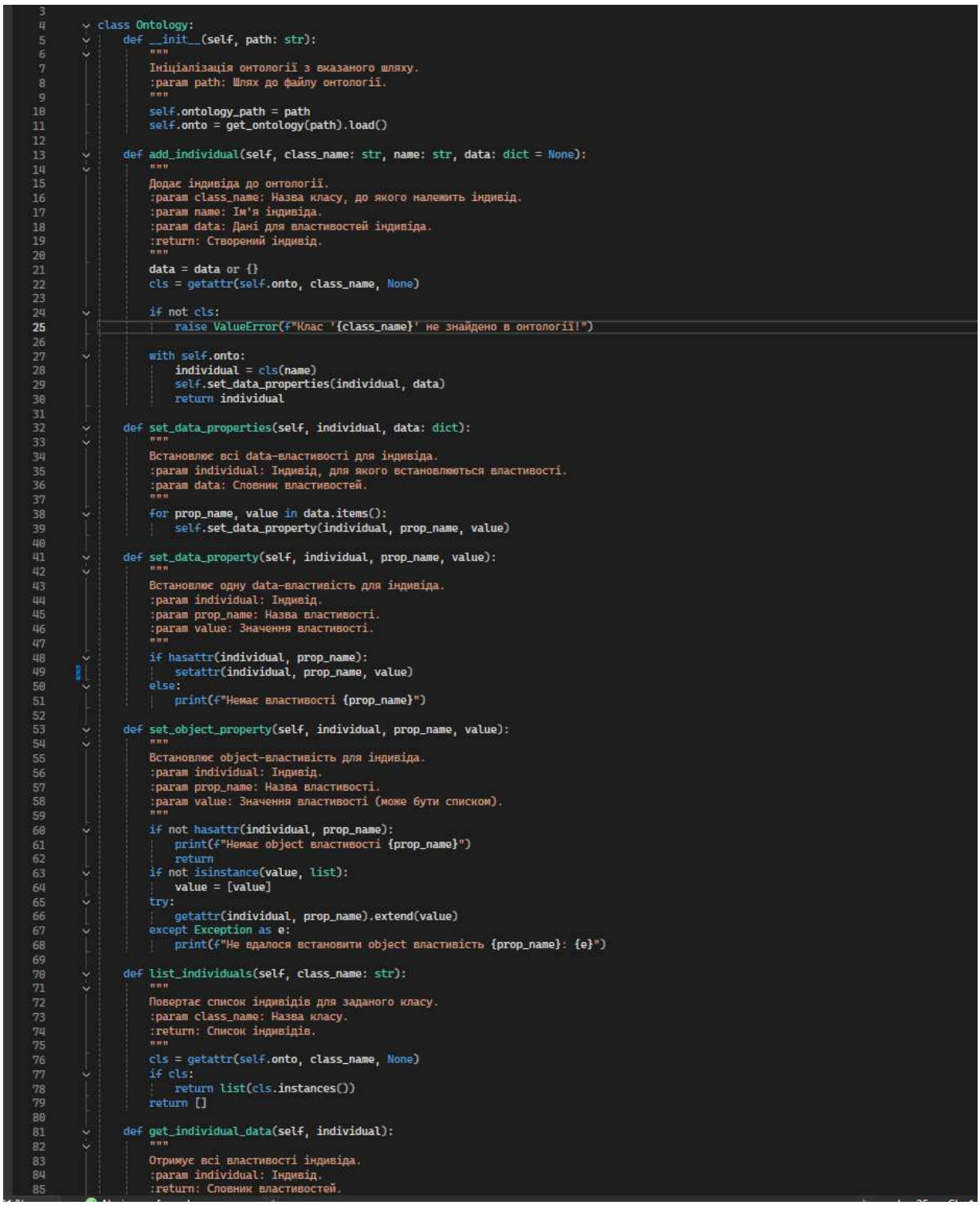
Код, що відповідає за графічний інтерфейс:

```python
class GeometrySolverGUI:
    def __init__(self, onto: Ontology, extractor: GeometryExtractor):
        self.onto = onto
        self.parser = extractor
        self.injector = OntologyInjector(self.onto)

        self.master = tk.Tk()
        self.master.title("Geometry Solver")

        self.label = tk.Label(self.master, text="Умова задачі:")
        self.label.pack()

        # Поле для відображення тексту задачі
        self.task_display = tk.Text(self.master, height=5, width=80, font=("Arial", 12))
        self.task_display.insert(tk.END, "Введіть задачу тут")
        self.task_display.config()
        self.task_display.pack(pady=10)

        self.run_button = tk.Button(self.master, text="Почати", command=self.solveButtonClick)
        self.run_button.pack(pady=10)

        self.step_frame = tk.Frame(self.master)
        self.step_frame.pack(pady=10)

        self.answer_entry = tk.Entry(self.step_frame, font=("Arial", 12), width=40)
        self.answer_entry.pack(pady=5)

        self.next_button = tk.Button(self.master, text="Далі", command=self.check_answer, state=tk.DISABLED)
        self.next_button.pack()

        # Додати випадайку з варіантами наступної дії
        self.action_var = tk.StringVar(self.master)
        self.action_var.set("Оберіть дію")  # Значення за замовчуванням

        self.action_menu = tk.OptionMenu(self.master, self.action_var, "Оберіть дію")
        self.action_menu.pack(pady=10)  # Додаємо випадайку до інтерфейсу

        self.history_label = tk.Label(self.master, text="Виконані кроки:")
        self.history_label.pack(pady=(20, 5))

        self.history_box = tk.Text(self.master, height=8, width=80, state=tk.DISABLED)
        self.history_box.pack()

        self.steps = []
        self.current_index = 0

        # Bind the Enter key to trigger the next_button's functionality
        self.master.bind("<Return>", self.trigger_next_button)

        self.master.mainloop()

    def solveButtonClick(self):
        task_text = self.task_display.get("1.0", tk.END).strip()

        self.steps.clear()
        self.current_index = 0
        self.next_button.config(state=tk.DISABLED)
        self.answer_entry.delete(0, tk.END)
        self.clear_history()

        geometry_model = self.parser.extract_chain(task_text)
        self.injector.inject(geometry_model)
        lines = ReasonerRunner.call_reasoning_api(self.onto.ontology_path, "b", "градусна_міра_кута")

        self.steps = []
        for line in lines:
            if '=' in line:
                step, answer = line.strip().split("=", 1)
                self.steps.append((step.strip(), answer.strip()))

        if self.steps:
            self.show_step()
            self.update_action_menu_with_steps()
```

```
    def show_step(self):
        self.answer_entry.delete(0, tk.END)

    def check_answer(self):
        _, correct_answer = self.steps[self.current_index]
        user_answer = self.answer_entry.get().strip()
        selected_action = self.action_var.get()

        # Перевірка, чи обрана дія відповідає очікуваному кроку
        expected_action = self.steps[self.current_index][0]  # Очікувана дія з кроку
        if selected_action.lower() not in expected_action.lower():
            messagebox.showerror("Невірна дія", f"Обрана дія '{selected_action}' не відповідає очікуваному кроку.")
            return

        if user_answer.lower() == correct_answer.lower():
            self.append_history(self.steps[self.current_index])
            self.current_index += 1
            if self.current_index < len(self.steps):
                self.show_step()
            else:
                messagebox.showinfo("Готово", "Ви успішно завершили задачу 🎉")
                self.answer_entry.delete(0, tk.END)
                self.next_button.config(state=tk.DISABLED)
        else:
            messagebox.showerror("Невірно", "Невірно. Спробуйте ще раз.")

    def paste_clipboard(self, event):
        try:
            clipboard = self.master.clipboard_get()
            self.text_input.insert(tk.INSERT, clipboard)
        except tk.TclError:
            pass
        return "break"

    def append_history(self, step_tuple):
        step_text, answer = step_tuple
        self.history_box.config(state=tk.NORMAL)
        self.history_box.insert(tk.END, f"{step_text} = {answer}\n")
        self.history_box.config(state=tk.DISABLED)

    def clear_history(self):
        self.history_box.config(state=tk.NORMAL)
        self.history_box.delete("1.0", tk.END)
        self.history_box.config(state=tk.DISABLED)

    def trigger_next_button(self, event):
        if self.next_button['state'] == tk.NORMAL:
            self.check_answer()

    def update_task_display(self, selected_task):
        """
        Оновлює текстове поле для відображення тексту задачі.
        :param selected_task: Обрана задача (ключ словника self.tasks).
        """
        self.task_display.config(state=tk.NORMAL)
        self.task_display.delete("1.0", tk.END)
        self.task = selected_task
        # Вставляємо текст задачі, відповідний вибраному ключу
        self.task_display.insert(tk.END, self.tasks.get(selected_task, ""))
        self.task_display.config(state=tk.DISABLED)

        # Оновлюємо варіанти дій у меню
        self.update_action_menu(selected_task)

    def update_action_menu_with_steps(self):
        """
        Оновлює варіанти в action_menu на основі self.steps.
        """
        menu = self.action_menu["menu"]
        menu.delete(0, "end")  # Очищаємо всі попередні варіанти

        for step, _ in self.steps:
            menu.add_command(label=step, command=lambda value=step: self.action_var.set(value))
```

No issues found

Код, що відповідає за взаємодію з онтологією:



```python
class Ontology:
    def __init__(self, path: str):
        """
        Ініціалізація онтології з вказаного шляху.
        :param path: Шлях до файлу онтології.
        """
        self.ontology_path = path
        self.onto = get_ontology(path).load()

    def add_individual(self, class_name: str, name: str, data: dict = None):
        """
        Додає індивіда до онтології.
        :param class_name: Назва класу, до якого належить індивід.
        :param name: Ім'я індивіда.
        :param data: Дані для властивостей індивіда.
        :return: Створений індивід.
        """
        data = data or {}
        cls = getattr(self.onto, class_name, None)

        if not cls:
            raise ValueError(f"Клас '{class_name}' не знайдено в онтології!")

        with self.onto:
            individual = cls(name)
            self.set_data_properties(individual, data)
            return individual

    def set_data_properties(self, individual, data: dict):
        """
        Встановлює всі data-властивості для індивіда.
        :param individual: Індивід, для якого встановлюються властивості.
        :param data: Словник властивостей.
        """
        for prop_name, value in data.items():
            self.set_data_property(individual, prop_name, value)

    def set_data_property(self, individual, prop_name, value):
        """
        Встановлює одну data-властивість для індивіда.
        :param individual: Індивід.
        :param prop_name: Назва властивості.
        :param value: Значення властивості.
        """
        if hasattr(individual, prop_name):
            setattr(individual, prop_name, value)
        else:
            print(f"Немає властивості {prop_name}")

    def set_object_property(self, individual, prop_name, value):
        """
        Встановлює object-властивість для індивіда.
        :param individual: Індивід.
        :param prop_name: Назва властивості.
        :param value: Значення властивості (може бути списком).
        """
        if not hasattr(individual, prop_name):
            print(f"Немає object властивості {prop_name}")
            return
        if not isinstance(value, list):
            value = [value]
        try:
            getattr(individual, prop_name).extend(value)
        except Exception as e:
            print(f"Не вдалося встановити object властивість {prop_name}: {e}")

    def list_individuals(self, class_name: str):
        """
        Повертає список індивідів для заданого класу.
        :param class_name: Назва класу.
        :return: Список індивідів.
        """
        cls = getattr(self.onto, class_name, None)
        if cls:
            return list(cls.instances())
        return []

    def get_individual_data(self, individual):
        """
        Отримує всі властивості індивіда.
        :param individual: Індивід.
        :return: Словник властивостей.
```

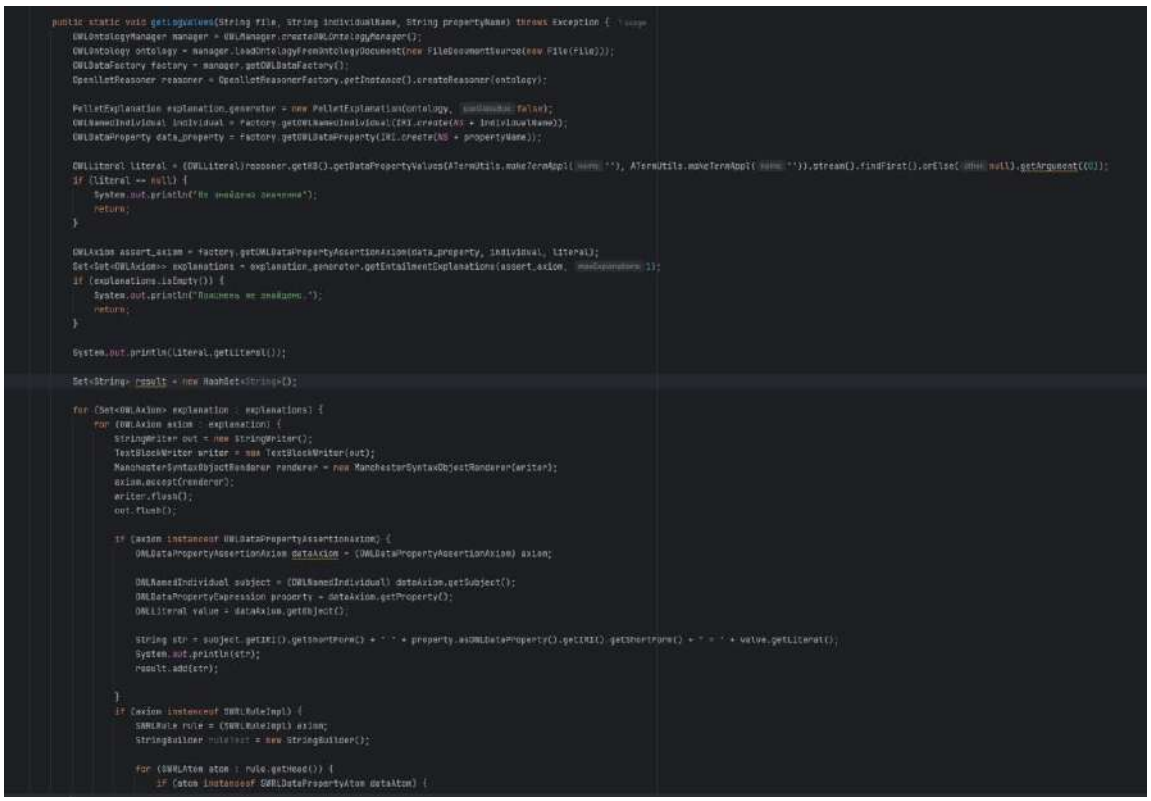
```python
class OntologyInjector:
    def __init__(self, ontology):
        """
        Ініціалізація інжектора для роботи з онтологією.
        :param ontology: Об'єкт онтології.
        """
        self.ontology = ontology
        self.counter = count()

    def add_properties_recursive(self, main_individual, properties, parent_name):
        """
        Рекурсивно додає властивості до індивіда.
        :param main_individual: Головний індивід.
        :param properties: Словник властивостей.
        :param parent_name: Ім'я батьківського індивіда.
        """
        for prop_name, values in properties.items():
            if not isinstance(values, list):
                self.ontology.set_data_property(main_individual, prop_name, values)
                continue

            for value in values:
                if isinstance(value, dict):  # object property
                    for cls_name, sub_props in value.items():
                        obj_individual_name = f"{parent_name}_{prop_name}_{next(self.counter)}"
                        obj_individual = self.ontology.add_individual(cls_name, obj_individual_name)

                        # Рекурсивно додаємо вкладені властивості
                        self.add_properties_recursive(obj_individual, sub_props, obj_individual_name)

                        self.ontology.set_object_property(main_individual, prop_name, obj_individual)
                else:  # data property
                    self.ontology.set_data_property(main_individual, prop_name, value)

    def inject(self, entity_property_results):
        """
        Інжектує дані в онтологію.
        :param entity_property_results: Список результатів з властивостями.
        """
        for i, descr in enumerate(entity_property_results, start=1):
            individual_name = f"{descr.individual}{i}"
            main_individual = self.ontology.add_individual(descr.individual, individual_name)
            self.add_properties_recursive(main_individual, descr.properties, individual_name)
```

Код, що відповідає за обробку природної мови:

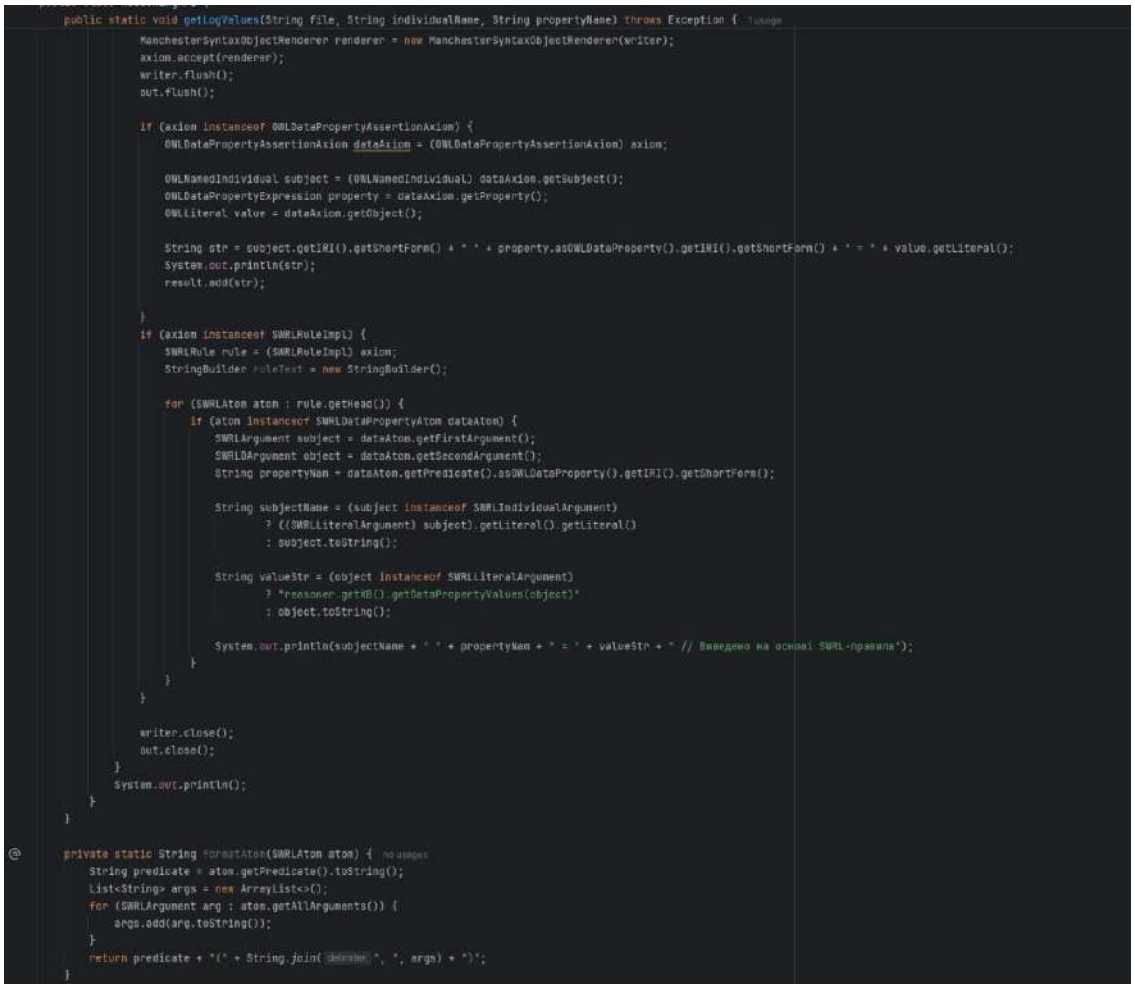
```python
from ufal.udpipe import Model, Pipeline

class EntityPropertyResult:
    def __init__(self, individual, properties):
        self.individual = individual
        # Ensure every value is a list
        self.properties = {
            prop: vals if isinstance(vals, list) else [vals]
            for prop, vals in properties.items()
        }

    def __repr__(self):
        return f"EntityPropertyResult(individual={self.individual}, properties={self.properties})"

class UdpipeParser:
    def __init__(self, model_path: str):
        self.model = Model.load(model_path)
        if not self.model:
            raise Exception("Не вдалося завантажити UDPipe модель")
        self.pipeline = Pipeline(self.model, "tokenize", "tag", "parse", "conllu")

    def analyze_text(self, text: str) -> str:
        result = self.pipeline.process(text)
        print(result)
        return result

    def parse_conllu(self, processed_text: str):
        sentences = []
        current_sentence = []
        for line in processed_text.strip().split('\n'):
            if line == '':
                if current_sentence:
                    sentences.append(current_sentence)
                    current_sentence = []
            elif not line.startswith('#'):
                parts = line.split('\t')
                if len(parts) >= 7:
                    token = {
                        'id': int(parts[0]),
                        'form': parts[1],
                        'lemma': parts[2],
                        'upostag': parts[3],
                        'head': int(parts[6]),
                        'deprel': parts[7]
                    }
                    current_sentence.append(token)
        if current_sentence:
            sentences.append(current_sentence)
        return sentences
```

```python
class GeometryExtractor:

    def __init__(self, model_path: str):
        self.parser = UdipipeParser(model_path)

    def extract_chain(self, text):
        parsed = self.parser.analyze_text(text)
        sentences = self.parser.parse_conllu(parsed)
        return self.extract_entity_property_value_chain(sentences)

    def extract_entity_property_value_chain(self, sentences):
        results = []

        # Визначення форм геометричних фігур
        shapes = {'трикутник': 'Трикутник', 'точка': 'Точка', 'коло': 'Коло', 'паралелограма': 'Паралелограм'}
        # Визначення властивостей фігур
        properties = {
            'площа': ('площа', 'data'),
            'периметр': ('периметр', 'data'),
            'радіус': ('радіус', 'data'),
            'катет': ('має_катет', 'object'),
            'катета': ('має_катет', 'object'),
            'вершина': ('має_точку', 'data'),
            'кут': ('кут', 'data'),
            'висота': ('angle', 'data'),
        }

        found_entity = None
        entity_properties = {}

        # Обробка кожного речення
        for sentence in sentences:
            # Пошук геометричної фігури
            found_entity = self._find_entity(sentence, shapes, found_entity)
            if not found_entity:
                continue

            # Витяг властивостей фігури
            self._extract_properties(sentence, properties, entity_properties)

        # Додавання результату, якщо знайдено фігуру
        if found_entity:
            results.append(EntityPropertyResult(individual=found_entity, properties=entity_properties))

        print(results)
        return results

    def _find_entity(self, sentence, shapes, found_entity):
        """
        Пошук геометричної фігури в реченні.
        :param sentence: список токенів речення
        :param shapes: словник форм фігур
        :param found_entity: вже знайдена фігура (якщо є)
        :return: знайдена фігура або None
        """
        for token in sentence:
            lemma = token['lemma'].lower()
            if not found_entity and lemma in shapes:
                return shapes[lemma]
        return found_entity

    def _extract_properties(self, sentence, properties, entity_properties):
        """
        Витяг властивостей фігури з речення.
        :param sentence: список токенів речення
        :param properties: словник властивостей
        :param entity_properties: словник властивостей фігури
        """
        for token in sentence:
            lemma = token['lemma'].lower()
            if lemma not in properties:
                continue

            prop_name, prop_type = properties[lemma]
            # Пошук числових значень, пов'язаних із властивістю
            value_tokens = [t for t in sentence if t['upostag'] == 'NUM']

            if value_tokens:
                for value_token in value_tokens:
                    self._process_value_token(value_token, prop_name, prop_type, entity_properties, sentence)
            else:
                # Додаємо властивість навіть якщо немає значення
                self._add_property_value(prop_name, prop_type, None, entity_properties)
```

Код на Java, що відповідає за логічне висновування:



```
public static void getLogValues(String file, String individualName, String propertyName) throws Exception {
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    OWLOntology ontology = manager.loadOntologyFromOntologyDocument(new FileDocumentSource(new File(file)));
    OWLDataFactory factory = manager.getOWLDataFactory();
    OpenlletReasoner reasoner = OpenlletReasonerFactory.getInstance().createReasoner(ontology);

    PelletExplanation explanation_generator = new PelletExplanation(ontology,  useUnsat: false);
    OWLNamedIndividual individual = factory.getOWLNamedIndividual(IRI.create(NS + individualName));
    OWLDataProperty data_property = factory.getOWLDataProperty(IRI.create(NS + propertyName));

    OWLLiteral literal = (OWLLiteral)reasoner.getKB().getDataPropertyValues(ATermUtils.makeTermAppl( term: ""), ATermUtils.makeTermAppl( term: "")).stream().findFirst().orElse( other: null).getArgument(0));
    if (literal == null) {
        System.out.println("Не знайдено значення");
        return;
    }

    OWLAxiom assert_axiom = factory.getOWLDataPropertyAssertionAxiom(data_property, individual, literal);
    Set<Set<OWLAxiom>> explanations = explanation_generator.getEntailmentExplanations(assert_axiom,  maxExplanations: 1);
    if (explanations.isEmpty()) {
        System.out.println("Пояснень не знайдено.");
        return;
    }

    System.out.println(literal.getLiteral());

    Set<String> result = new HashSet<String>();

    for (Set<OWLAxiom> explanation : explanations) {
        for (OWLAxiom axiom : explanation) {
            StringWriter out = new StringWriter();
            TextBlockWriter writer = new TextBlockWriter(out);
            ManchesterSyntaxObjectRenderer renderer = new ManchesterSyntaxObjectRenderer(writer);
            axiom.accept(renderer);
            writer.flush();
            out.flush();

            if (axiom instanceof OWLDataPropertyAssertionAxiom) {
                OWLDataPropertyAssertionAxiom dataAxiom = (OWLDataPropertyAssertionAxiom) axiom;

                OWLNamedIndividual subject = (OWLNamedIndividual) dataAxiom.getSubject();
                OWLDataPropertyExpression property = dataAxiom.getProperty();
                OWLLiteral value = dataAxiom.getObject();

                String str = subject.getIRI().getShortForm() + " " + property.asOWLDataProperty().getIRI().getShortForm() + " = " + value.getLiteral();
                System.out.println(str);
                result.add(str);

            }
            if (axiom instanceof SWRLRuleImpl) {
                SWRLRule rule = (SWRLRuleImpl) axiom;
                StringBuilder ruleText = new StringBuilder();

                for (SWRLAtom atom : rule.getHead()) {
                    if (atom instanceof SWRLDataPropertyAtom dataAtom) {
```



```
public static void getLogValues(String file, String individualName, String propertyName) throws Exception {
            ManchesterSyntaxObjectRenderer renderer = new ManchesterSyntaxObjectRenderer(writer);
            axiom.accept(renderer);
            writer.flush();
            out.flush();

            if (axiom instanceof OWLDataPropertyAssertionAxiom) {
                OWLDataPropertyAssertionAxiom dataAxiom = (OWLDataPropertyAssertionAxiom) axiom;

                OWLNamedIndividual subject = (OWLNamedIndividual) dataAxiom.getSubject();
                OWLDataPropertyExpression property = dataAxiom.getProperty();
                OWLLiteral value = dataAxiom.getObject();

                String str = subject.getIRI().getShortForm() + " " + property.asOWLDataProperty().getIRI().getShortForm() + " = " + value.getLiteral();
                System.out.println(str);
                result.add(str);

            }
            if (axiom instanceof SWRLRuleImpl) {
                SWRLRule rule = (SWRLRuleImpl) axiom;
                StringBuilder ruleText = new StringBuilder();

                for (SWRLAtom atom : rule.getHead()) {
                    if (atom instanceof SWRLDataPropertyAtom dataAtom) {
                        SWRLArgument subject = dataAtom.getFirstArgument();
                        SWRLDArgument object = dataAtom.getSecondArgument();
                        String propertyNam = dataAtom.getPredicate().asOWLDataProperty().getIRI().getShortForm();

                        String subjectName = (subject instanceof SWRLIndividualArgument)
                                ? ((SWRLLiteralArgument) subject).getLiteral().getLiteral()
                                : subject.toString();

                        String valueStr = (object instanceof SWRLLiteralArgument)
                                ? "reasoner.getKB().getDataPropertyValues(object)"
                                : object.toString();

                        System.out.println(subjectName + " " + propertyNam + " = " + valueStr + " // Виведено на основі SWRL-правила");
                    }
                }
            }

            writer.close();
            out.close();
        }
        System.out.println();
    }
}

private static String formatAtom(SWRLAtom atom) {
    String predicate = atom.getPredicate().toString();
    List<String> args = new ArrayList<>();
    for (SWRLArgument arg : atom.getAllArguments()) {
        args.add(arg.toString());
    }
    return predicate + "(" + String.join( delimiter: ", ", args) + ")";
}
```

Код, що відповідає за виклик Reasoning API:

```python
class ReasonerRunner:
    def call_reasoning_api(owl_file_path, individual, property_name):
        cmd = [
            "java", "-cp",
            "lib/GEOM-1.0-SNAPSHOT-jar-with-dependencies.jar",
            "org.example.ReasoningAPI",
            owl_file_path,
            individual,
            property_name
        ]

        process = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
        stdout, stderr = process.communicate()

        if process.returncode != 0:
            print("Java process error:\n", stderr)
            return None

        print("Java output:\n", stdout)

        # опціонально: витягни тільки результати
        values = []
        for line in stdout.splitlines():
            values.append(line.strip())
        return values
```