Лабораторна робота №3

Виконала: студентка групи ПКН-2
Пілецька Єлізавета

11 травня 2025 р.

1 Вступ

Метою даної лабораторної роботи було імплементувати клас, який компілює скінченний автомат на основі заданого регулярного виразу та використовує цей автомат для перевірки відповідності рядків. Було реалізовано розпізнавання певних літер, чисел та також метасимволів ‘.‘, ‘*‘ та ‘+‘. Робота виконана в рамках вивчення патерну.

2 Реалізація класів станів

Було доповнено наданий каркас коду, реалізувавши логіку для кожного конкретного стану. Основою є абстрактний клас State(ABC).

2.1 Клас `State`

Абстрактний базовий клас, що визначає інтерфейс для всіх станів. Він вимагає реалізації методів init та checkself. Важливою деталлю є ініціалізація списку наступних станів self.nextstates = [] в конструкторі кожного конкретного стану для уникнення спільного використання цього списку між екземплярами.

2.2 Клас `StartState`

Представляє початковий стан автомата. Конструктор init викликає конструктор базового класу. Метод checkself у наданій користувачем реалізації викликає абстрактний метод базового класу. Зазвичай, StartState не обробляє символи безпосередньо, а слугує точкою входу, тому checkself для нього міг би повертати False.

2.3 Клас `TerminationState`

Приймаючий стан.

- `__init__()`: Ініціалізує цей стан.
- `check_self(self, char: str) -> bool`: Не споживає символи, позначає успішне завершення розпізнавання. Тому і завжди повертає хибу (або False).

2.4 Клас `DotState`

Представляє метасимвол ‘.‘.

- `__init__()`: Ініціалізує стан.
- `check_self(self, char: str) -> bool`: Цей метод завжди повертає ‘True‘.

2.5 Клас `AsciiState`

Призначений для розпізнавання конкретних літер чи цифр.

- `__init__(self, symbol: str)`: Зберігає символ, який цей стан має розпізнавати, в атрибуті екземпляра self.currsym.
- `check_self(self, curr_char: str) -> bool`: Порівнює вхідний символ currchar зі збереженим self.currsym.

2.6 Клас `StarState`

Представляє оператор ‘*‘ (нуль або більше входжень).

- `__init__(self, checking_state: State)`: Зберігає стан checkingstate, який відповідає символу або групі, що має повторюватися.
- `check_self(self, char: str) -> bool`: У реалізації цей метод перевіряє, чи може один з наступних станів обробити поточний символ. Це відрізняється від перевірки символу, що повторюється.

2.7 Клас `PlusState`

Представляє оператор ‘+‘.

- `__init__(self, checking_state: State)`: Аналогічно до StarState, зберігає checkingstate.
- `check_self(self, char: str) -> bool`: У реалізації користувача цей метод повертає False. Вказує, що сам стан PlusState слугує для організації переходів у графі автомата.

3 Реалізація класу `RegexFSM`

Цей клас відповідає за компіляцію регулярного виразу та перевірку рядків.

3.1 Конструктор `__init__(self, regex_expr: str)`

Метод будує скінченний автомат, обробляючи вираз посимвольно. Для операторів ‘*‘ та ‘+‘ використовується механізм ніби підглядання на наступний символ.

- **Літерали та ‘.‘**: Створюються відповідні стани, що додаються до ланцюга.
- **Обробка ‘*‘**: Створюється стан для ‘X‘, а також спеціальний стан, який з’єднується з попередніми станами через перехід.
- **Обробка ‘+‘**: Створюється стан для ‘X‘ та ‘PlusState‘, з прямим переходом , щоб забезпечити хоча б одне входження.
- В кінці додається ϵ-перехід.

3.2 Метод `__init_next_state(...)`

Метод, що створює відповідний стан на основі токена. Логіка для ‘*‘ та ‘+‘ обробляється в конструкторі, а метод створює стани для символів перед операторами.

3.3 Метод `get_closure(self, states: set[State]) -> set[State]`

Обчислює ϵ-замикання для заданої множини станів:

1. Початкова множина додається до результату та черги.
2. Черга обробляється в ширину.

Цей метод є ключовим для роботи НСА.

3.4 Метод `check_string(self, text_to_check: str) -> bool`

Симулює роботу НСА для перевірки рядка:

1. Ініціалізується множина активних станів.
2. Для кожного символу рядка формуються нові потенційні стани.
3. Якщо після всіх символів є хоча б один ‘TerminationState‘, рядок вважається відповідним.

Метод починається з ϵ-замикання початкового стану.

4 Висновок

Зреалізовано клас ‘RegexFSM‘, що компілює регулярний вираз в скінченний автомат та перевіряє рядки на відповідність. Використано методи побудови автомату.