Звіт до лабораторної роботи: Скінченний автомат (FSM) для співставлення регулярних виразів

Яблуновська Анастасія

2025 рік

1 Вступ

У цій лабораторній роботі реалізовано скінченний автомат (FSM) для співставлення регулярних виразів (regex). Основною метою є створення базового FSM, який розпізнає:

- Літери англійського алфавіту (як великі, так і малі)
- Цифри
- Спеціальні символи: `*` (нуль або більше повторень) та `+` (одне або більше повторень).

2 Реалізовані класи

FSM для співставлення регулярних виразів містить кілька ключових станів:

- **StartState**: Початковий стан FSM, з якого починається обробка.
- **TerminationState**: Останній стан, що позначає успішне завершення обробки, якщо введена строка відповідає шаблону.
- **DotState**: Стан для символу `.` у regex, який може відповідати будь-якому символу.
- **AsciiState**: Стан для окремих ASCII символів.

- **StarState**: Стан для символу `*`, що дозволяє повторення попереднього стану нуль або більше разів.
- **PlusState**: Стан для символу `+`, що вимагає, щоб попередній стан повторювався хоча б один раз.

Кожен стан відповідає за перевірку, чи підходить поточний символ для цього стану, і за перехід до наступного стану, якщо символ підходить.

3 Реалізація

3.1 Клас State

Клас `State` є абстрактним базовим класом для всіх станів FSM. Він визначає:

- Абстрактний метод `check_self`, який перевіряє, чи підходить заданий символ для поточного стану.
- Метод `check_next`, який перевіряє переходи між станами залежно від наступного символу.

3.2 StartState

`StartState` — це початковий стан FSM. Він не обробляє жодних символів, але є точкою входу для автомата. Перехід з цього стану залежить від першого символу в регулярному виразі.

3.3 TerminationState

`TerminationState` — це останній стан у FSM, який позначає, що введена строка відповідає регулярному виразу. Цей стан не обробляє жодних символів, тому метод `check_self` повертає `False`.

3.4 DotState

`DotState` відповідає за символ `.` у регулярному виразі. Цей символ може відповідати будь-якому символу, тому метод `check_self` в цьому стані завжди повертає `True`, що означає, що він приймає будь-який символ.

3.5 AsciiState

`AsciiState` представляє окремі ASCII символи. Конструктор цього стану ініціалізує його конкретним символом, а метод `check_self` порівнює поточний символ з цим символом.

3.6 StarState та PlusState

Для `StarState` і `PlusState` дозволено повторення попереднього стану. Для `StarState` це може бути нуль або більше разів, а для `PlusState` — хоча б один раз.

3.7 Конструктор RegexFSM

Конструктор класу `RegexFSM` приймає регулярний вираз і компілює його в набір станів. Він проходить через кожен символ регулярного виразу, ініціалізуючи відповідний стан для кожного символу і зв'язуючи їх між собою.

3.7.1 Метод `check_string`

Метод `check_string` реалізує логіку перевірки рядка на відповідність регулярному виразу. Він перевіряє, чи входить введений рядок в заданий шаблон регулярного виразу, використовуючи механізм скінченного автомата (FSM).

Алгоритм роботи

1. **Активні стани**: Початково автомат знаходиться в одному стані — поточному стартовому стані. Цей стан позначено як `active_states`, що є множиною активних станів автомата.

2. **Перевірка кожного символу рядка**: Для кожного символу з введеної строки алгоритм перевіряє, які стани можуть бути активними після його обробки. Для цього створюється нова множина `next_active_states`, куди додаватимуться стани, до яких можна перейти на основі поточного символу.

3. **Обробка кожного активного стану**: Для кожного активного стану алгоритм перевіряє, чи можна перейти в інші стани. Перехід залежить від типу стану:

- Для стану `StarState` (символ `*` у регулярному виразі), який дозволяє повторення попереднього елемента, можуть бути кілька варіантів переходів: перехід до наступних станів або самоперехід (повторення того ж самого стану).

- Для стану `PlusState` (символ `+`), що вимагає хоча б одного повторення попереднього елемента, здійснюється перехід тільки тоді, коли символ підходить до перевіряючого стану.

- Інші стани (наприклад, `DotState` для символу `.` або `AsciiState` для конкретних символів) просто перевіряють, чи відповідає поточний символ у рядку їх вимогам.

4. **Перехід до наступного символу**: Після перевірки одного символу всі наступні активні стани оновлюються на основі результатів цієї перевірки. Якщо на певному етапі немає активних станів, це означає, що введений рядок не відповідає регулярному виразу, і метод повертає `False`.

5. **Перевірка на успішне завершення**: Наприкінці, якщо після обробки всіх символів у рядку залишаються активні стани, то перевіряється, чи серед них є фінальний стан `TerminationState`, що вказує на завершення перевірки.

Метод використовує стек для обробки складних переходів і забезпечує правильну роботу регулярних виразів з операціями повторення.