Звіт до лабораторної роботи №3

Оксана Москв'як

Вступ

Основним завданням було реалізувати свій "regex" використовуючи знання по темі скінченні автомати, а саме в даній реалізації недетермінований скінченний автомат (НСА), та дослідити як можна власноруч створювати перевірку для певних патернів.

Реалізовані класи

Було реалізовано наступні класи:

Абстрактний клас `ABCState`

Абстрактний клас, який є основою всіх станів. Зберігає наступні стани та розпочинає перевірку того, чи стан може розпізнати певний символ.

Клас `StartState`

Клас, який є початком НСА (Q_0). У ньому реалізовано лише перевірку `check_self()`, яка завжди дає результат `False`, оскільки початковий стан не може приймати ніякий символ.

Клас `TerminationState`

Клас, який обробляє кінцевий стан. Цей стан не приймає жодних символів, тому `check_self()` завжди повертає `False`. Саме він визначає чи рядок приймається автоматом чи ні.

Клас `DotState`

Обробляє символ `'.'` з регулярного виразу, суть якого в визначенні будь-якого символу 1 раз. Саме тому імплементована функція `check_self()` завжди повертає `True`.

Клас `AsciiState`

Клас, який обробляє ASCII-символи, що вказані в регулярному виразі. У методі `check_self()` відбувається перевірка, чи символ, що подається на вхід, збігається з очікуваним.

Клас `StarState`

Клас, що обробляє символ `'*'` з регулярного виразу, який означає, що попередній елемент може зустрічатися нуль або більше разів. Під час ініціалізації `StarState` зберігає посилання як на стан, до якого оператор застосований, так і на самого себе у списку `next_states`, даючи можливість повернення та перевикористання цього символу.

Клас `PlusState`

Клас, що реалізує поведінку оператора `'+'` у регулярному виразі. Символ `'+'` вказує, що попередній елемент має зустрітися один або більше разів. Як і в класі `StarState`, під час ініціалізації `PlusState` зберігає посилання на стан, до якого оператор застосований, і додає як цей стан, так і самого себе до списку `next_states`. Завдяки цьому забезпечується можливість повернення до символів.

Клас `RegexFSM`

Основа програми. У класі реалізовано побудову скінченного недетермінованого автомату та обробку стрічок, які приходять на перевірку.

Метод `__init__`

У конструкторі `__init__` відбувається основна логіка – побудова автомату на основі regex стрічки. Обробляється випадок, коли стрічка починається із спеціальних символів `'+'` або `'*'`, адже на такому етапі немає попереднього елемента для повторення.

Наступним кроком обробляється пара символів: поточний символ та наступний, якщо він є спеціальним символом (`'+'` або `'*'`) і не є крапкою (`'.'`). Це відповідає обробці переходу стану в `StarState` або `PlusState`. Під час цієї перевірки створюється поточний стан, який далі передається в `StarState` або `PlusState` для збереження символу, який ми хочемо відтворити декілька разів. Оскільки розглядається пара значень, індекс проходження по стрічці збільшується на 2.

Ще однією перевіркою є обробка пари символів, де наступний символ дорівнює `'+'`. Це відповідає обробці переходу стану в `PlusState`. Оскільки `'+'` може працювати лише коли є точно одне входження символу перед ним, то саме на цьому етапі обробляється створення самого стану і вже можливе його повторення у `PlusState`. У цій перевірці також розглядається пара значень, і аналогічно індекс збільшується на 2. Основною імплементацією є створення нового стану, яке супроводжується передачею його у список наступних станів попереднього стану та в допоміжний список `all_states`. Поточний стан оновлюється після ітерації, стаючи новим попереднім станом, а індекс проходження по стрічці збільшується на 1.

Метод `__init_next_state()`

Дана функція "призначає" стан до якогось із символів, що надходить, відповідно даючи йому стан/символ чи просто створюючи для подальшого використання.

Метод `check_string()`

Функція виконує перевірку, чи заданий рядок відповідає побудованому автомату для регулярного виразу. Відбувається ітерація по рядку та зберігання поточних допустимих станів. Для кожного символу з рядка перевіряється, чи може хоча б один із наступних станів обробити його методом `check_self()`. Якщо так, то цей стан додається до `next_states`, і саме вони стають новим списком `current_states`. Після обробки всіх символів рядка функція перевіряє, чи принаймні один із поточних станів переходить до `TerminationState` – кінцевого стану автомата, та видає відповідний результат `True` або `False`.

Висновок

Головною метою цієї лабораторної роботи було дослідження патерну програмування State і, зокрема, використати знання, здобуті під час вивчення теми автоматів. Проте моя реалізація `RegexFSM` не є повною через певні труднощі, які виникли в мене в процесі створення лабораторної роботи, але це був гарний досвід роботи з автоматами.