Звіт з лабораторної роботи
Фінітний автомат для регулярного виразу

Ія Магарита

11 травня 2025 р.

Нам було запропоновано реалізувати свій власний фінітний автомат для простої перевірки стрічок на відповідність регулярному виразу на Python. Для цього нам надали заготовку коду і інструкції.

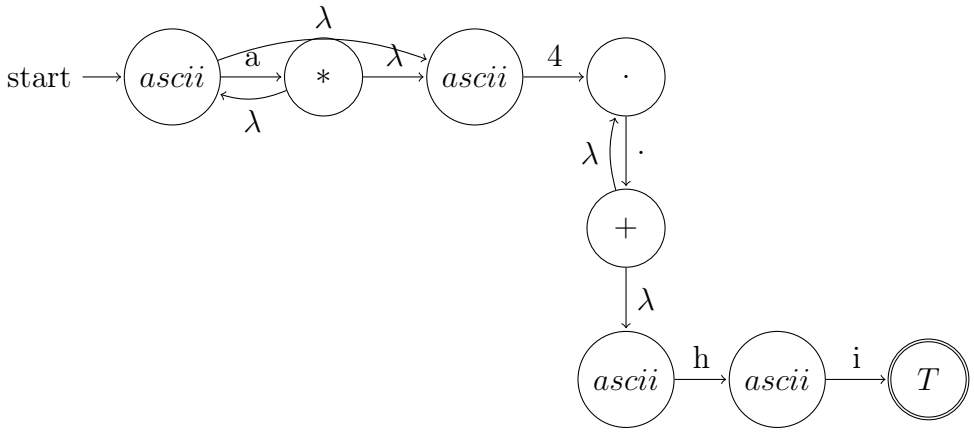
FSA

Фінітний автомат (finite state automaton, FSA) — це модель обчислення, яка обробляє послідовність символів (наприклад, текстовий рядок), переходячи між кінцевою кількістю станів, залежно від прочитаних символів. Його використовують для перевірки, чи певний рядок відповідає заданому шаблону (наприклад, регулярному виразу).

Як приклад, нам надали такий регулярний вираз:

`a*4.+hi`

Графічно цей вираз я бачу як:



Якщо словами, то цей вираз описує мову, в якій:

- передує 0 або більше літер `a`,
- далі — символ `4`,

- один або більше будь-яких символів (оператор `.` з квантифікатором `+`),
- після чого — символи `h` та `i` у кінці.

Орієнтуючись на такий тип побудови, можна приступити до написання коду.

Опис реалізації

У роботі реалізовано класи для кожного з можливих станів автомату:

- `StartState` — початковий стан;
- `AsciiState` — для перевірки конкретних символів;
- `DotState` — приймає будь-який символ;
- `StarState` — квантифікатор `*` (нуль або більше входжень);
- `PlusState` — квантифікатор `+` (одне або більше входжень);
- `TerminationState` — фінальний стан.

Під час побудови автомату рядок регулярного виразу парситься посимвольно, та для кожного символу створюється відповідний стан, з'єднаний переходами через поле `next_states`.
Більше того! для правильної обробки $*$ та $+$ я реалізувала читання одного символу попереду. Тобто FSA одразу знає, чи символ іде просто як частина виразу, чи в парі з якимось іншим прапорцем.

0.1 `RegexFSM.__init__`

Тут вибудовується ланцюг регулярної мови.
Читається кожен символ рядку, що задає мову. Поступово створюється кожен обєкт класу.
Для звичайних символів без додаткових операторів нічого особливого не відбувається. Вони просто створюються, а в `next_states` потім додається наступний обєкт-стан.
Що ж до символів, що ідуть на пару з $+$ і $*$, то в них структура складніша.

Спершу створюється стан, який має повторюватись.
Потім ми створюємо стан `PlusState` або `StarState`, куди ми першим ділом передаємо в `self.state` попередній стан, як `checking_state`.
Якщо ми знаходимось в `StarState`, то в `next_states` ми передаємо цей повторювальний стан, який також має посилання на себе. Далі, при нарощенні дерева, `StarState.next_states` буде поповнюватись і *при першій перевірці на валідність першочергово в `next_states` повторювального стану запишуться всі стани*, якими було поповнено `StarState.next_states`.
Після того вже відбувається перевірка, чи хоч один із наступних станів у „Зірочці"може прийняти символ.
В `PlusState` логіка схожа, але ми початково не записуємо `checking_state` в `next_states`.
При першій перевірці ми першочергово *наповнюємо посиланнями перевіряючий стан*,

а після того залишаємо список посилань в „Плюсі“порожнім. Тоді ми дивимось, чи перевіряючий стан здатен прийняти символ рядка. Це забезпечує, що перевіряючий стан буде задоволений хоча б раз, на відміну від стану в „Зірочці“.
В кінці ми також додаємо `TerminationState`.

`check_self` і `check_next`

Тож, тепер у нас є створений автомат, і ми можемо перейти до побудови перевірки стрічок.
Кожен стан має в собі `check_self` та `check_next`. Ці дві ф-ці викликаються в `check_string`, себто в перевірці рядка.
`check_self` перевіряє, чи на даному етапі символ відповідає умові стану, в якому ми знаходимось. Ця ф-ція викликаєтсья безпосередньо в `check_next`. І якщо все окей - ми передаємо стан, в якому все валідно відбулось як новий стан. при цьому це може бути той самий стан - якшо в нас присутня петля.
Також, в `PlusState` ця ф-ція працює по-іншому - вона викликає `check_next` свого `self.state`. Така вже особливість в „Плюса“.
А в `DotState` ми перевіряємо наступні стани з кінця, щоб забезпечити, що програма не буде читати всі наступні символи як „будь-що“.

Висновки

Загалом, програма доволі примітивна і не охоплює увесь функціонал RegEx, але базові фічі присутні та працюють. В будь-якому разі, її розробка дала краще розуміння скінченних автоматів, особливостей їх використання і важливості нормального сну(бо без нього я б її не зробила).