Звіт з лабораторної роботи
Фінітний автомат для регулярного виразу

Ія Магарита

10 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.

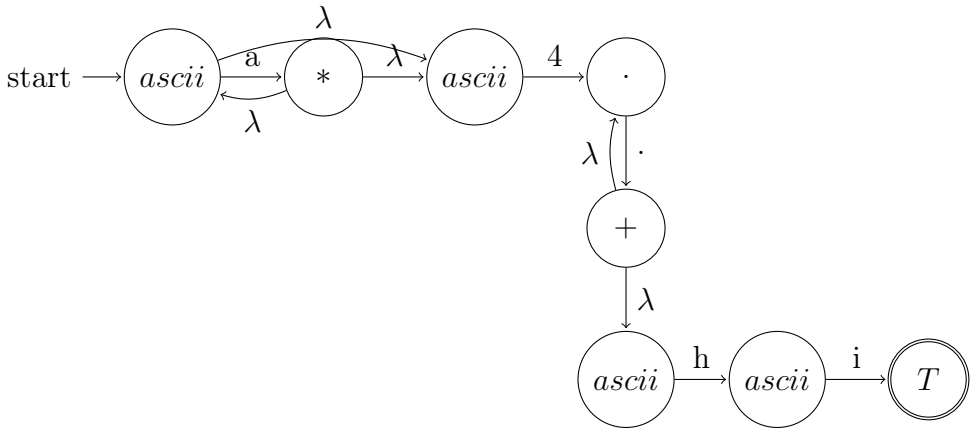
FSA

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 додаємо його ще в `next_states`. Тоді ми наступним елементом в `next_states` передаємо `StarState`, що посилається на повторюваний символ. Це зроблено якраз для забезпечення повторення.
В `StarState`, що містить в собі такий же `StarState` все просто, бо всі стани будуть додаватись в `next_states` у одну й ту ж комірку памяті та завжди будуть там залишатись, навіть при повторенні. А от з `PlusState` це не так легко було реалізувати, адже ми одразу не знаємо, який елемент регулярного виразу йтиме потім, і отже не можемо у внутрнішній `StarState` передати жоден наступний стан безпосередньо.

Тому це було забезпечено у `check_next`. Всі стани з `PlusState.next_states`, які йдуть після `StarState`, при першій же перевірці передаються у нього. І тому повторення працює!
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 теперішній умові або одній з наступних. Якщо все окей - ми переходимо в наступний стан з допомогою `check_next`.
Особливість цієї ф-ції в тому, що коли `check_self` перевіряє список станів з початку, ця перевіряє його з кінця. Це побудовано так тому, що в `check_self` нам пріоритетніше, аби символ відповідав теперішньому станові, а в `check_next` ми хочемо знайти наступний стан, який не дотичний до теперішнього, а такі стани завжди будуть в кінці. І якщо символ відповідає станові, на який ми маємо посилання із теперішнього - то ф-уія повертає його.
Більше того, без цього обертання програма зациклиться - вона постійно буде в стані „Зірочка“, бо той завжди матиме в кінці посилання на наступний, і навіть якщо символ не відповідає теперішньому станові, програма буде всеодно крутитись навколо нього, адже вважатиме валідним. Тому так.

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).