Звіт з лабораторної роботи

Владислав Данилишин

11 травня 2025 р.

У цій лабораторній роботі реалізовано FSA (finite state automaton), який приймає регулярні вирази з простими модифікаторами ($*$, $+$, `.`) і перевіряє рядки на відповідність. В основі — об'єктно-орієнтована архітектура з базовим класом `State` та спадкоємцями для кожного типу стану.

Мета

Побудувати такий автомат, який на основі регулярного виразу може:

- Створити зв'язану між собою послідовність станів,
- Проітеруватись по рядку і верифікувати його структуру,
- Вивести результат верифікації рядка залежно від ітерації по ньому.

Класи станів

Кожен стан — окремий клас. Всі вони наслідують абстрактний базовий клас `State`, в якому оголошено обов'язкові методи `check_self` та `check_next`:

- `check_self` — перевіряє, чи символ підходить до поточного стану.
- `check_next` — перебирає `next_states`, перевіряючи відповідність наступного символу.

Далі — огляд конкретних станів:

- **StartState** — початковий. Має лише список `next_states`, решта реалізується через спадкування.
- **AsciiState** — приймає конкретний символ (наприклад, `h` чи `4`).
- **DotState** — завжди повертає `True`, тобто приймає будь-який символ.
- **StarState** — реалізація оператора $*$ (нуль або більше символів).
- **PlusState** — реалізація оператора $+$ (одне або більше символів).
- **TerminationState** — output стан.

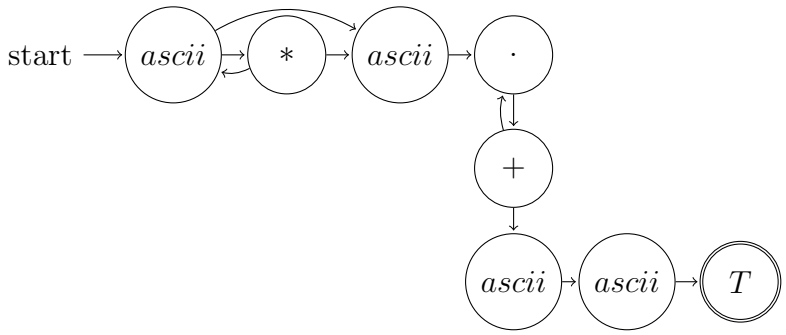
Побудова автомату

Конструктор класу `RegexFSM` приймає регулярний вираз у вигляді рядка. Подальший процес обробки:

1. Створюється початковий стан, від якого будемо надбудовувати стани.
2. Проходимось по символах регулярного виразу:
   - Якщо наступний символ це $*$ або $+$, створюється відповідна пара: символ $+$ стан, що буде накручувати цей символ.
   - Інакше — створюється окремий `AsciiState` чи `DotState`.
3. Для кожного нового стану формуються переходи через `next_states`.
4. Вкінці додається `TerminationState` — ознака успішного завершення.

Реалізація $+$ та $*$ не суттєво різниця. Якщо є символ та $+$, то ми спокійно заходимо спочатку у `AsciiState`, а потім у **PlusState**. Проте $*$ треба робити інакше. Потрібно аби спочатку курсор заходив у **StarState**, а потім уже до перевірочного символа, при його наявності у стрінгу, що треба перевірити. (Реалізація присутня у конструкторі класу `RegexFSM`)

Автомат, що поданий як приклад у файлі, виглядає так:



Перевірка рядка

Метод `check_string` класу `RegexFSM` виконує посимвольну перевірку:

- Ітеруємось по даному рядку.
- Пробуємо знайти стан у `next_states` поточного, який може обробити цей символ.
- Якщо знайдено — переходимо до нього, інакше автомат не зчитує стрінг, і тому буде виведено `False`.
- Якщо доходимо до кінця — перевіряємо, чи серед наступних станів є `TerminationState`. Є - повертаємо `True`, ні - `False`