Лабораторна робота 3

Максим Головін

1 Вступ

Програма є реалізацією скінченного автомата - вона приймає спочатку регулярний вираз на основі якого створює послідовність станів, щоби потім перевіряти можливість прийняття довільної стрічки.

2 Код

2.1 Стани

1.State - клас від якого успадковуються інші класи.
2.StartState - початковий стан атвомата, в який входить стрічка.
3.TerminationState - кінцевий стан автомата. Його досягнення означає, що рядок прийнятий автоматом.
4.DotState - стан, який відповідає '.' в регулярному виразі. Приймає будь-який символ.
5.AsciiState - приймає ASCII символи.
6.StarState - стан, який відповідає '*'. Може повторювати попередній символ нуль або більше разів.
7.PlusState - стан, який відповідає '+'. Може повторювати попередній символ один або більше разів.

2.2 RegexFSM

Клас для створення послідовності станів, і для перевірки, чи може атвомат прийняти стрічку. 1.У функція init і init next state на основі наданого регулярного виразу створюється послідовність станів.

2.2.1 Створення послідовності станів

1.Якщо символ регулярного виразу є ні '+' ні '*', то ми створюємо новий стан, додаємо його до наступних станів минулого стану і проводимо заміну де минулий стан прирівнюємо до теперішнього, а супер минулий стан до минулого.

```
tmp_next_state = self.__init_next_state(char, prev_state,
    tmp_next_state)

prev_state.next_states.append(tmp_next_state)
if super_prev is not None:
    super_prev.next_states.append(tmp_next_state)
super_prev = prev_state
prev_state = tmp_next_state
```

2.Якщо символ регулярного виразу це '+', то ми так само як і до цього створюмо новий стан, і додаємо його до наступних станів минулого стану, після чого робимо заміни. Але крім цього ми створюємо зациклення, так щоби символ перед +, зустрічався один або більше разів у стрічці.

В init:

```
tmp_next_state = self.__init_next_state(char, prev_state,
    tmp_next_state)
prev_state.next_states.append(tmp_next_state)
super_prev = prev_state
prev_state = tmp_next_state
```

В init next state:

```
new_state = PlusState(tmp_next_state)
tmp_next_state.next_states.append(tmp_next_state)
```

3.Якщо символ регулярного виразу це '*' то ми так само як і до цього створюмо новий стан, і додаємо його до наступних станів минулого стану, але замін не робимо. Як і в стані '+', ми робимо зациклення, але щоби символ перед '*' міг зустрічатись і нуль разів, то коли ми переходимо до наступного символу після '*' і створюємо стан для нього, то ми додаємо цей новий стан до наступних станів не тільки минулого стану, а й супер минулого стану, який відстає від теперішнього на два кроки. Наприклад, в нас є стрічка 'a*4.+hi'. Коли після '*', ми переходимо до '4' і створюємо стан для нього, то додаємо '4' як наступний стан для 'a'. Так би це працювало для '+', але так як символ перед '*' може зустрітись і нуль разів, то ми додаємо '4' як новий стан ще і для початкового стану.

В init:

```
tmp_next_state = self.__init_next_state(char, prev_state,
    tmp_next_state)
prev_state.next_states.append(tmp_next_state)
```

В init next state:

```
new_state = StarState(tmp_next_state)
tmp_next_state.next_states.append(tmp_next_state)
```

2.2.2 check string

У функції check string проходить перевірка чи може автомат прийняти стрічку.
Спочатку визначається список possible next states, в котрому на початку буде StartState. Потім цикл проходиться по кожному символу, і перевіряє чи наступні стани теперішнього стану можуть прийняти цей символ. Якщо так, то ці стани додаються до new possible next states, котрі потім передаються як значення для possible next states перед новою ітерацією. І так до кінця циклу. В кінці перевіряється чи є TerminationState в possible next states. Якщо так, то автомат може прийняти стрічку. Якщо ні, то не може.

```
        our_state = self.curr_state
        possible_next_states = [our_state]
        for char in string:
            new_possible_next_states = []

            for state in possible_next_states:
                if isinstance(state, TerminationState):
                    break
                for s_state in state.next_states:
                    if s_state.check_self(char):
                        if s_state not in
                            new_possible_next_states:
                            new_possible_next_states.append(
                                s_state)
            possible_next_states = new_possible_next_states


        for other_state in possible_next_states:
            if isinstance(other_state, TerminationState):
                return True
        return False
```